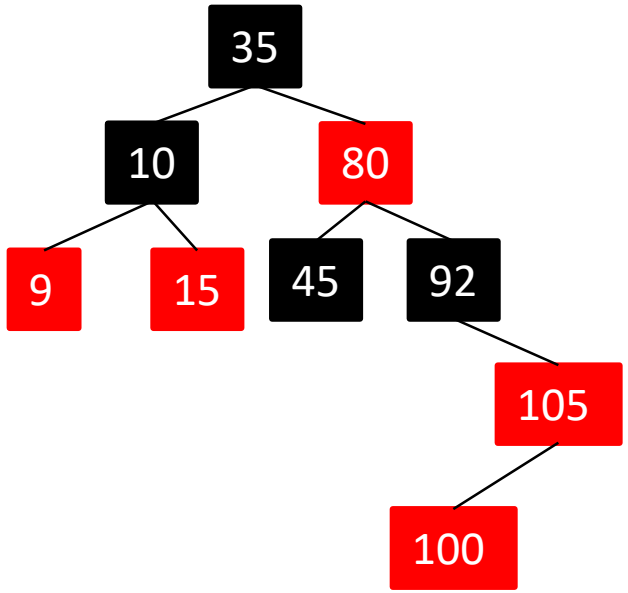
Sessió 21 Arbres

LP 2019-20

Red Black Tree. Inserció: ROTACIÓ

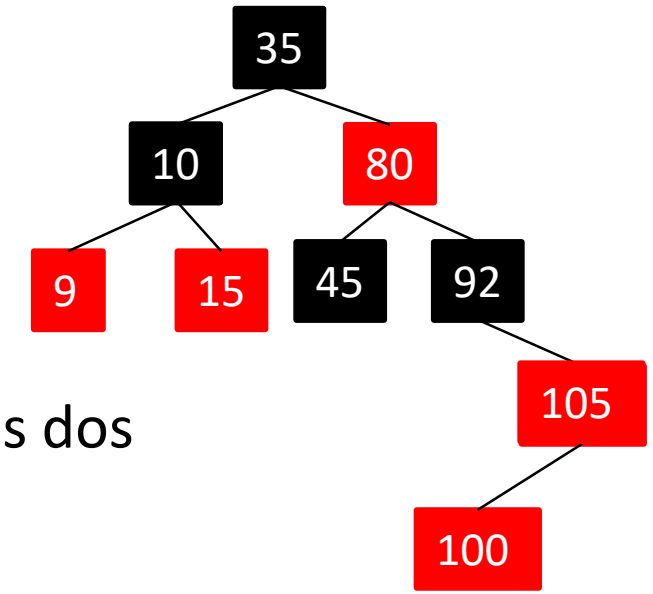
Tornem a l'aplicació web: https://tommikaikkonen.github.io/rbtree/#

- Insereix els nodes 35, 15, 80, 45, 92, 10, 9, 105, 100



Red Black Tree. Inserció: ROTACIÓ

Tornem a l'aplicació web



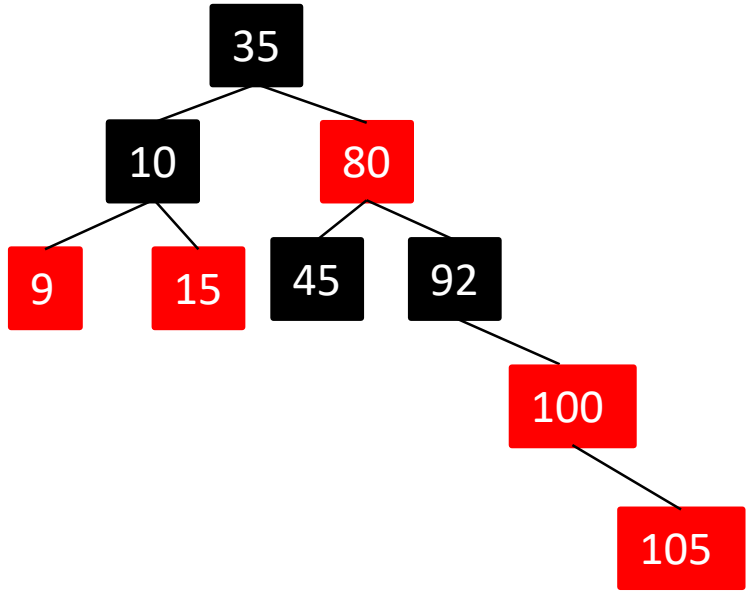
- Quan inserim el node 100, què passa?

  El fill i el pare són de costats diferents no són els dos esquerre o els dos dret

  Podem passar a que els dos siguin del mateix costat?

Red Black Tree. Inserció: ROTACIÓ

Tornem a l'aplicació web



- Quan inserim el node 100, què passa?

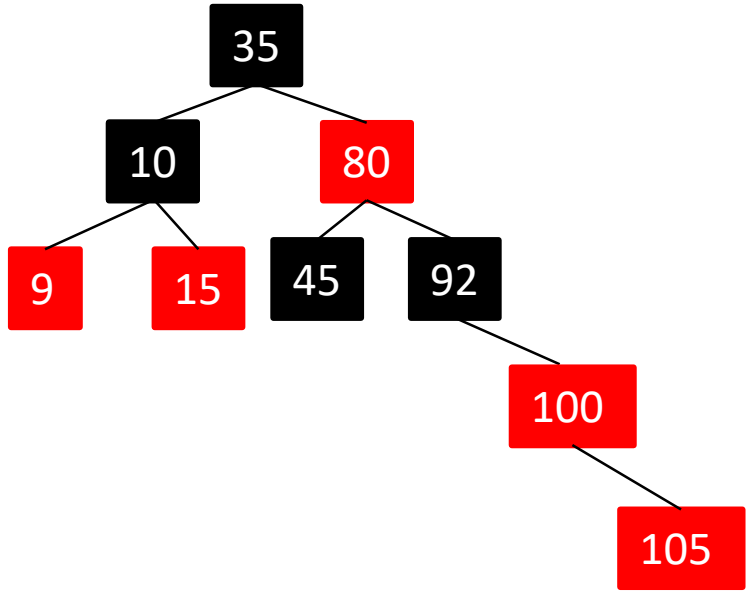
  El fill i el pare són de costats diferents no són els dos esquerre o els dos dret

  Podem passar a que els dos siguin del mateix costat?

  Rotem pare (105) a la dreta i obtenim així el cas dret dret: sabeu resoldre'l?

Red Black Tree. Inserció: ROTACIÓ

Tornem a l'aplicació web



- Quan inserim el node 100, què passa?

  El fill i el pare són de costats diferents no són els dos esquerre o els dos dret

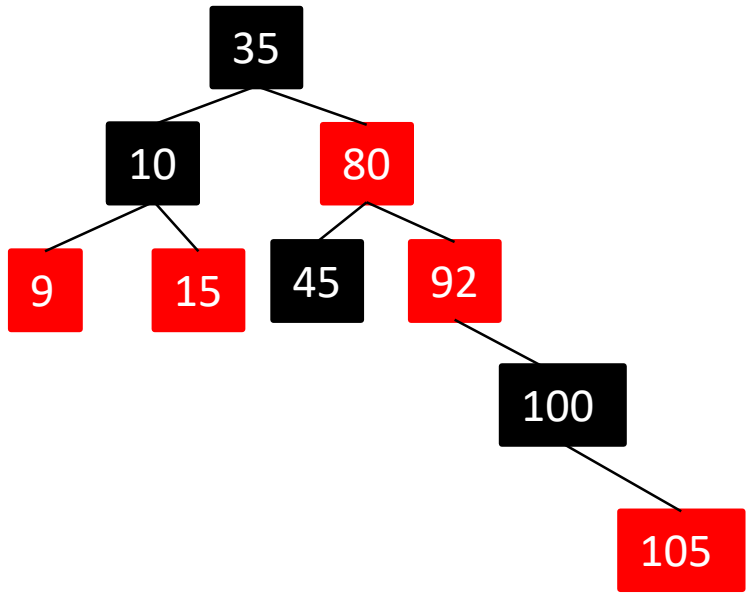
  Podem passar a que els dos siguin del mateix costat?

  Rotem pare (105) a la dreta i obtenim així el cas dret dret: sabeu resoldre'l?
  És el simètric del cas esquerre-esquerre

Red Black Tree. Inserció: ROTACIÓ

Tornem a l'aplicació web

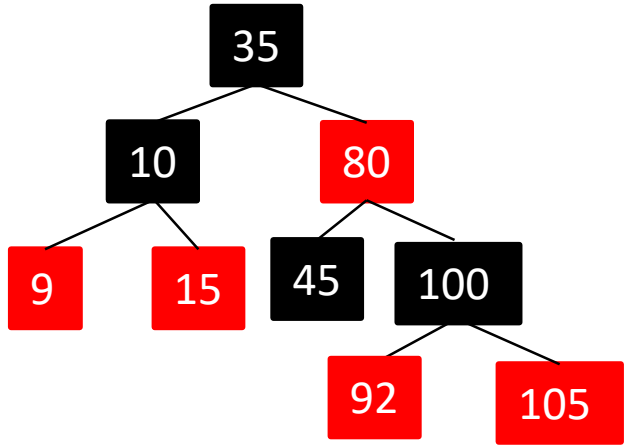
- Quan inserim el node 100, què passa?



Ja tenim el cas dret dret:
a) intercanviem colors pare i avi

Red Black Tree. Inserció: ROTACIÓ

Tornem a l'aplicació web



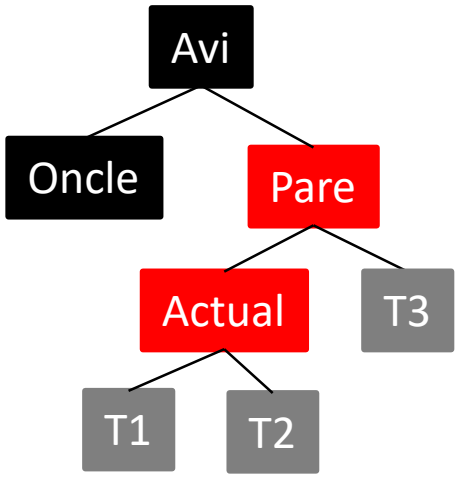
- Quan inserim el node 100, què passa?

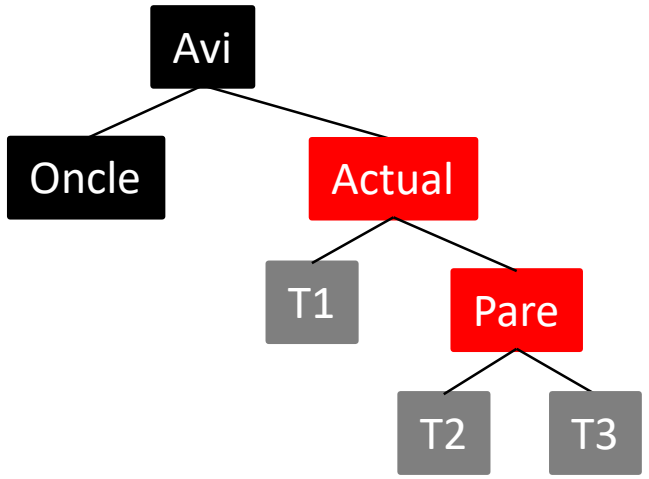
Ja tenim el cas dret dret:

a) intercanviem colors pare i avi
b) Rotació a l'esquerra des de l'avi

Red Black Tree. Rotacions

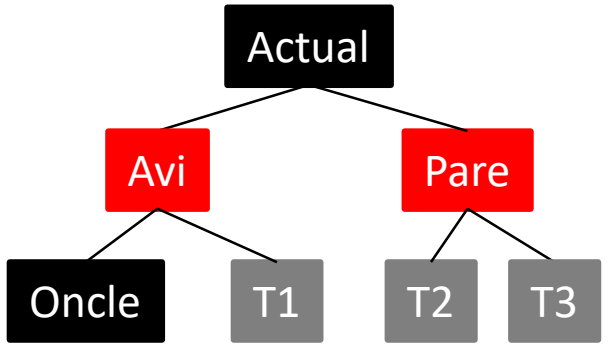
2. CAS: Dret Esquerre: El pare de node actual és fill dret i Node actual és fill esquerre





Rotació a dreta Pare

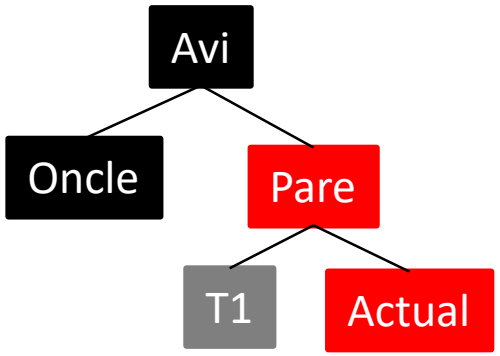
Cas Dret Dret

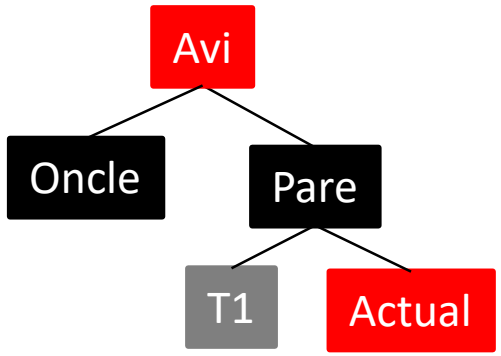


Resoldre Cas Dret Dret
com a simètric d'esquerre-esquerre

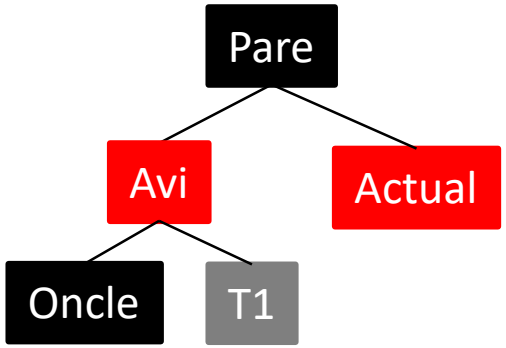
Red Black Tree. Rotacions

3. CAS: Dret Dret: Node actual és fill dret i el seu pare també





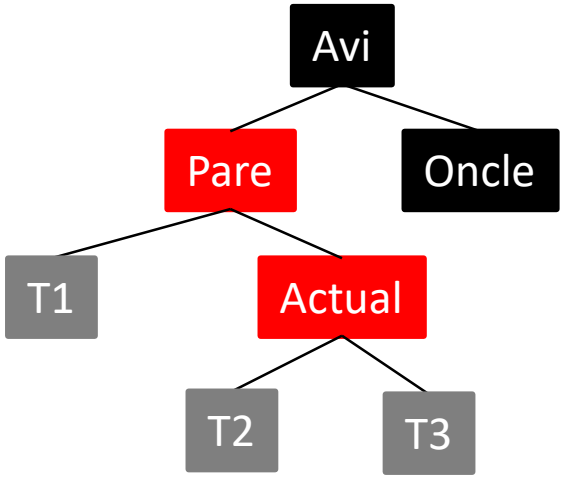
Canvi Color Pare →Avi

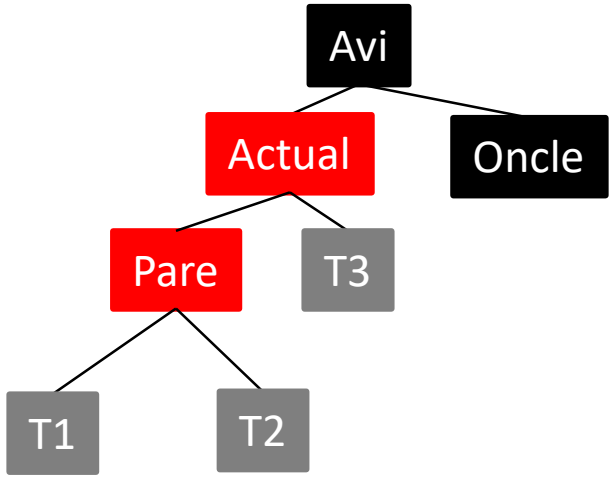


Rotació a Esquerra Avi

Red Black Tree. Rotacions

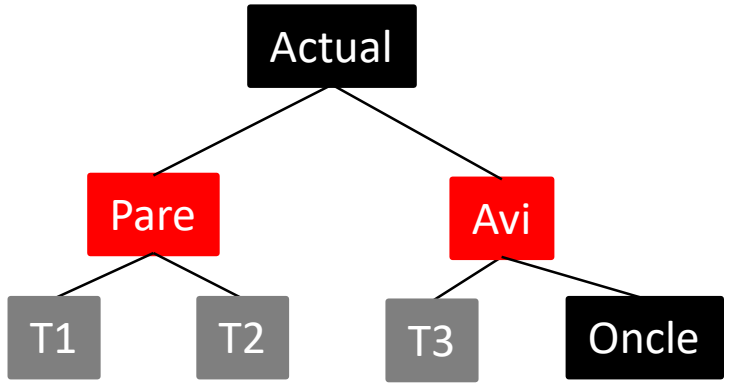
4. CAS: Esquerre dret: El pare de node actual és fill esquerre i Node actual és fill dret





Rotació a Esquerra Pare

Cas Esquerre-Esquerre

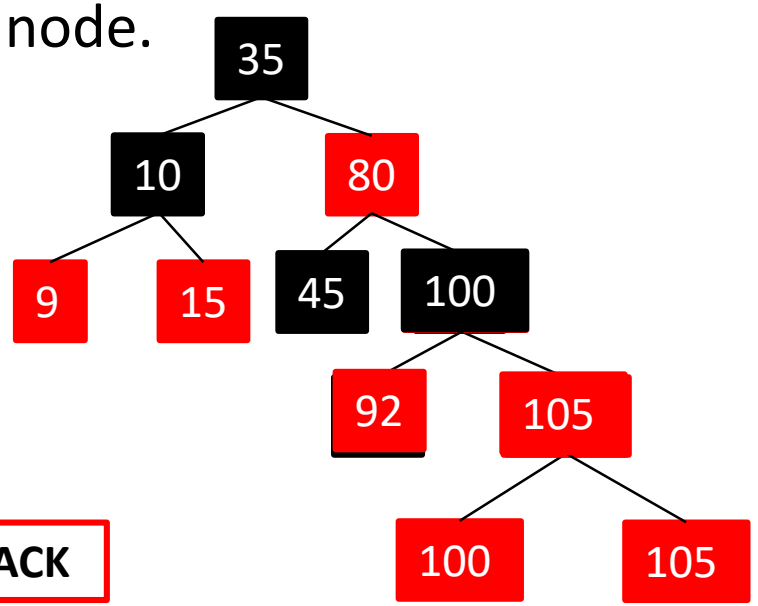


Resoldre Cas Esquerre-Esquerre

Red Black Tree. Inserció: ROTACIÓ

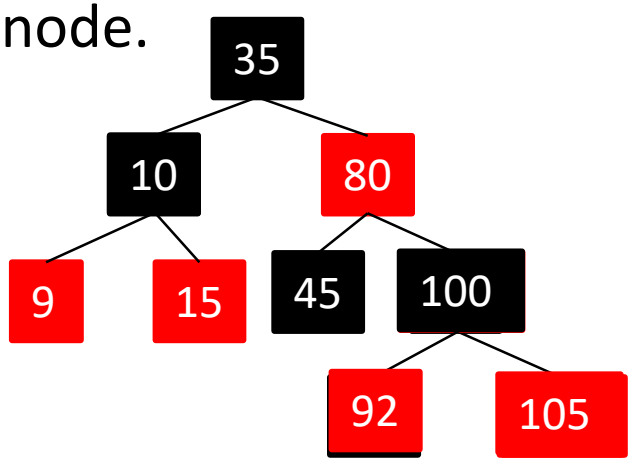
Inserció: 100

1. Utilitzem la cerca binària per saber 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 Dret-Esquerre**
                - **a. Rotació a dreta 105**
                - **b. Dret-Dret**
                    - **a. Swap color 92-100**
                    - **b. Rotació a esquerra 92**



Red Black Tree. Inserció: ROTACIÓ

Inserció: 100



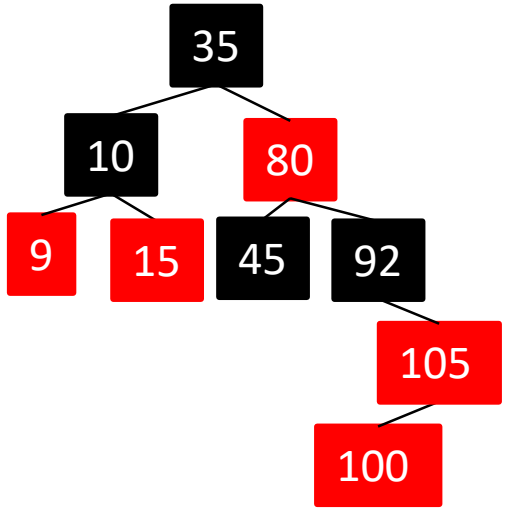
1. Utilitzem la cerca binària per saber on va el nou node.
2. Un node nou ***(x)*** és **RED**.
3. Si ***(x)*** és l'arrel el posem a **BLACK**
4. Sino
    1. si el pare és **BLACK** ja està
    2. Sino (el pare és **RED)**
        1. Si l'oncle és **RED** (avi ha de ser **BLACK**)
        2. Si oncle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

NULL es considera com a **BLACK**

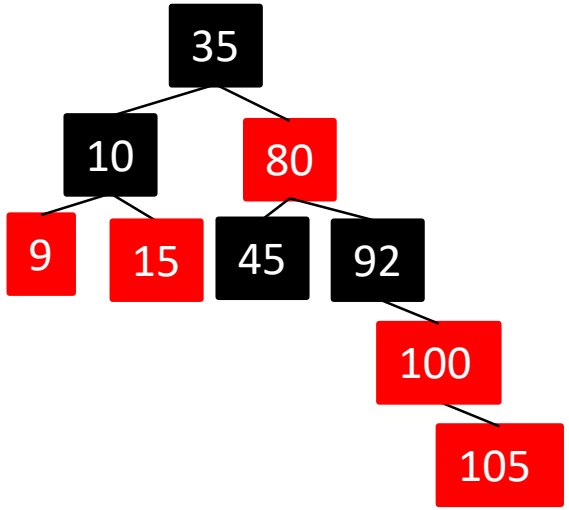
        i. **Cas Dret-Esquerre**
            a. **Rotació a dreta 105**
            b. **Dret-Dret**
                a. **Swap color 92-100**
                b. **Rotació a esquerra 92**

Red Black Tree. Inserció: ROTACIÓ

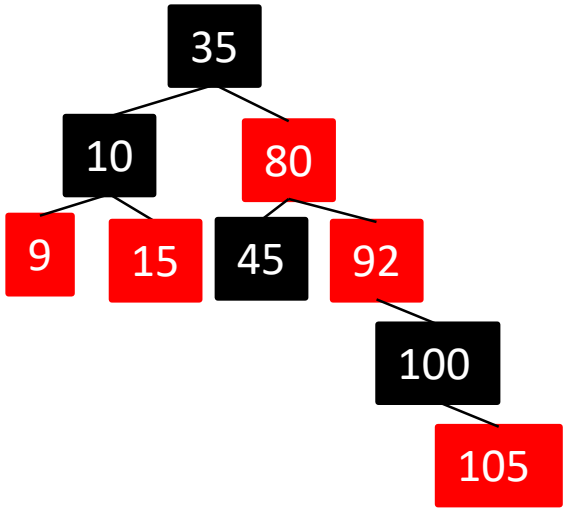
Inserció: 100



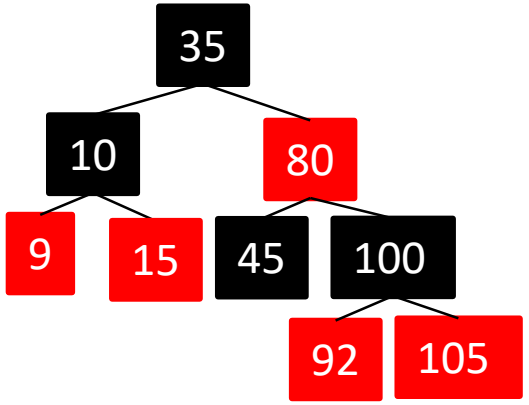
1. Cerca binària.
2. Inserim vermell



3. Pare vermell, oncle negre o null
4. Rotació cas dret-esquerre
   4-1. Rotació a dreta pare



4.2 cas dret-dret
4.2.1. canvi color 100-92

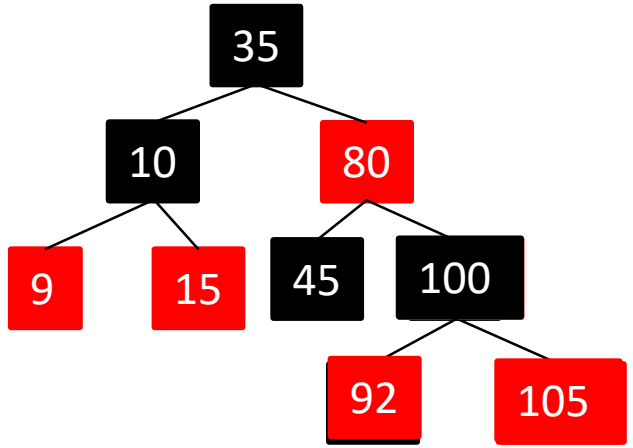


4.2.2 rotació a esquerra avi

Red Black Tree. Inserció: ROTACIÓ

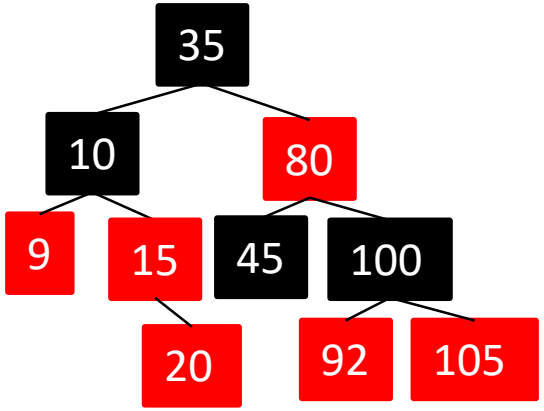
Tornem a l'aplicació web

- Insereix el node 20

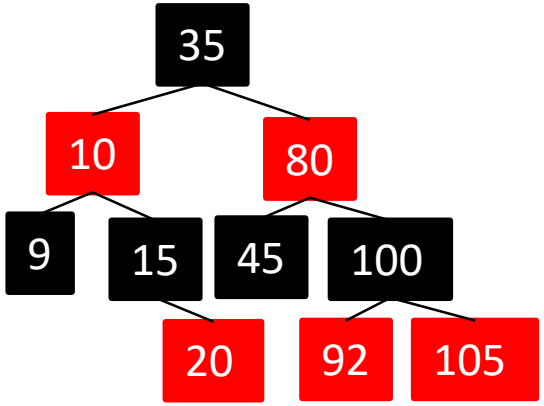


Red Black Tree. Inserció: ROTACIÓ

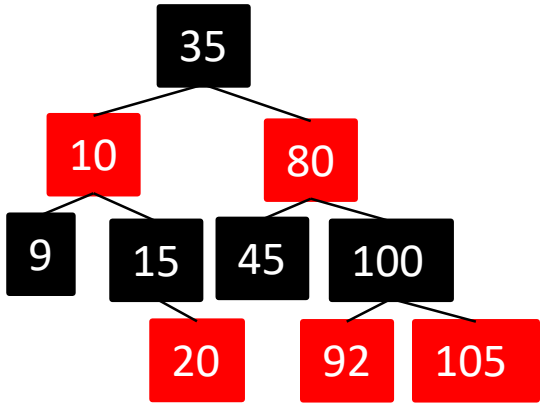
Inserció: 20



1. Cerca binària.
2. Inserim vermell



3. Pare i oncle vermells→ els posem negres
4. Avi negre el posem vermell

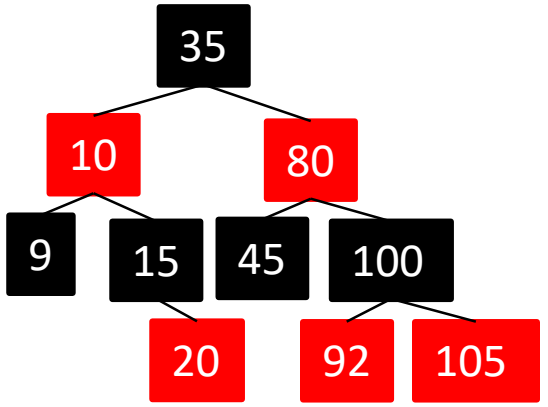


5. Mirem de forma recursiva avi
6. Si pare és negre ja està

Red Black Tree. Inserció: ROTACIÓ

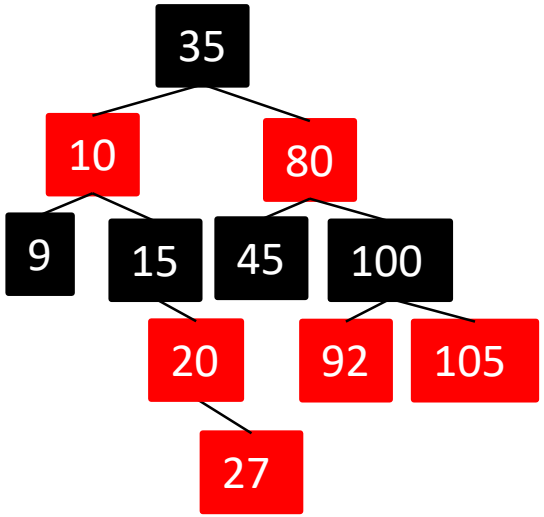
Tornem a l'aplicació web

- Insereix el node 27

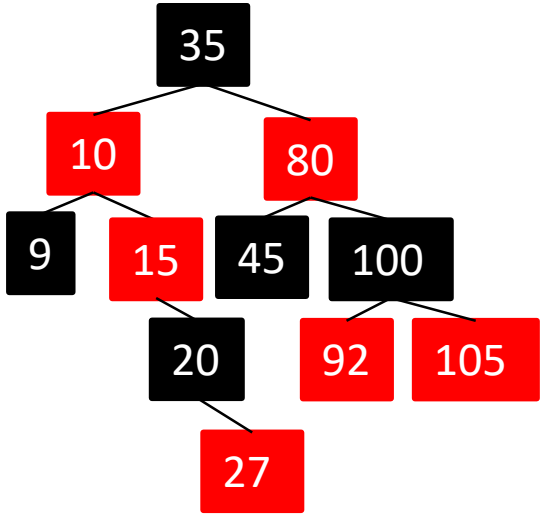


Red Black Tree. Inserció: ROTACIÓ (Cas Dret-Dret)

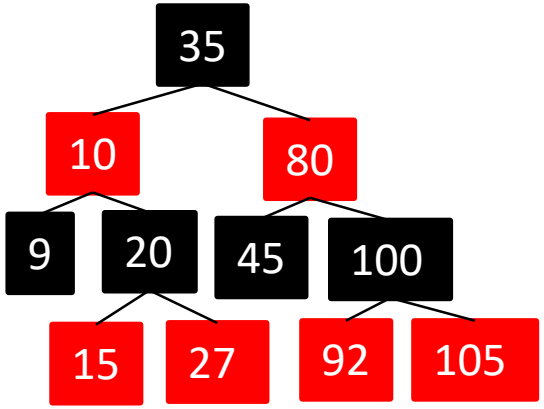
Inserció: 27



1. Cerca binària.
2. Inserim vermell



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



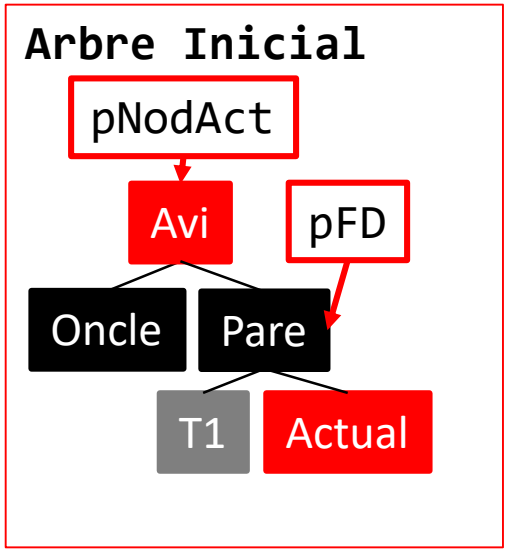
4.2 Rotació a esquerra avi

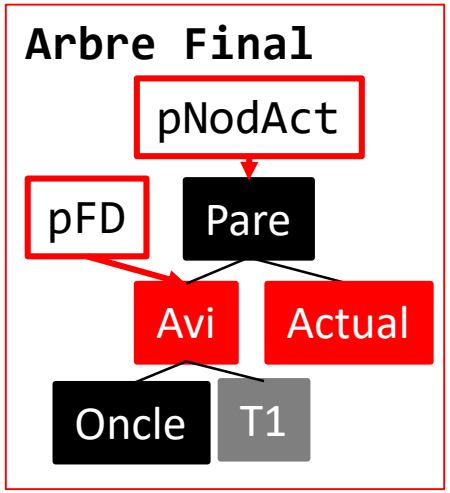
Exercici

Implementa:

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

Penseu que al fer rotacions podem estar canviant el node arrel. Ho haurem de tenir en compte i en cas que l'avi sigui arrel farem l'actualització de punters mitjançant el mètode `swapRootContents( pNodAct, pFD );`



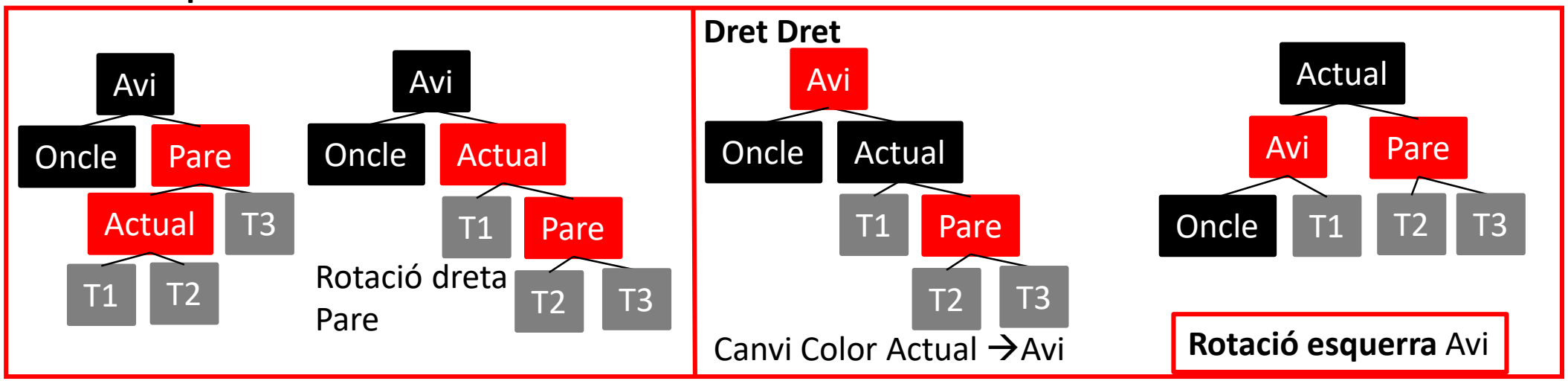


Exercici

Modifica mètode rota pel cas dret-esquerre i dret-dret:

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

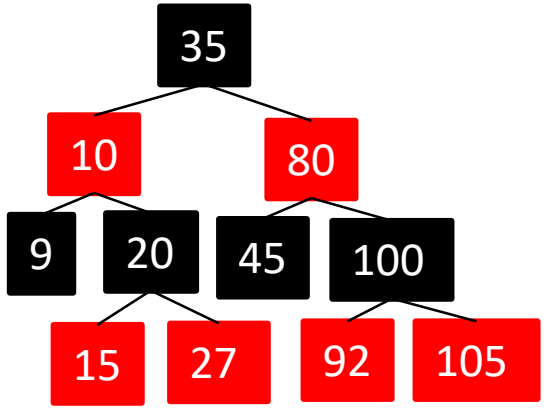
Cas Dret-Esquerre



Nota: Cas dret-Esquerre és igual que dret-dret amb un pas prèvi de rotació dreta del pare.

Red Black Tree. Inserció: ROTACIÓ

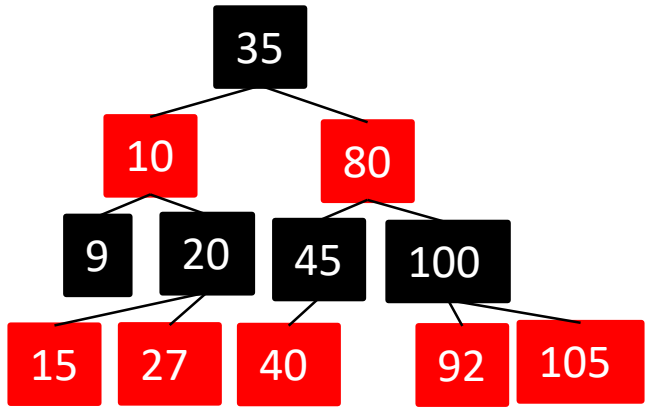
Tornem a l'aplicació web



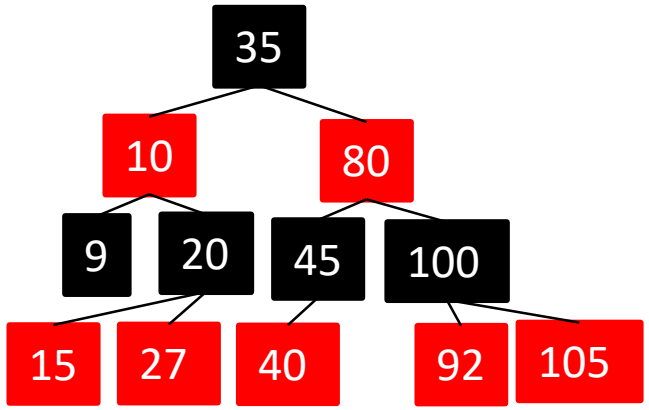
- Insereix el node 40
- Insereix el node 43

Inserció: Red Black Tree.

Inserció: 40



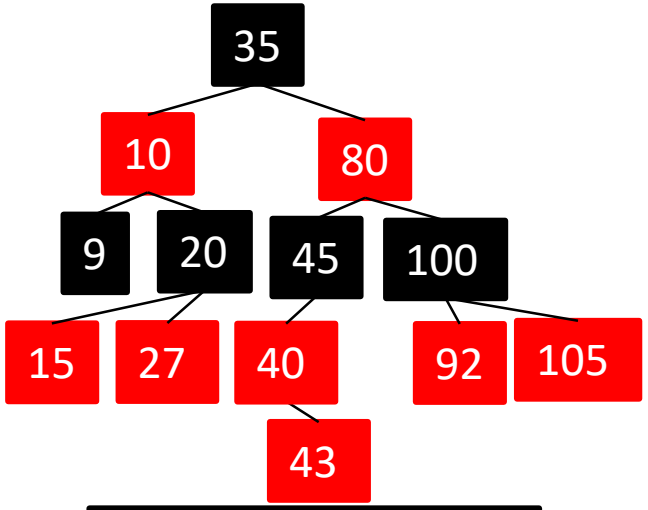
1. Cerca binària.
2. Inserim vermell



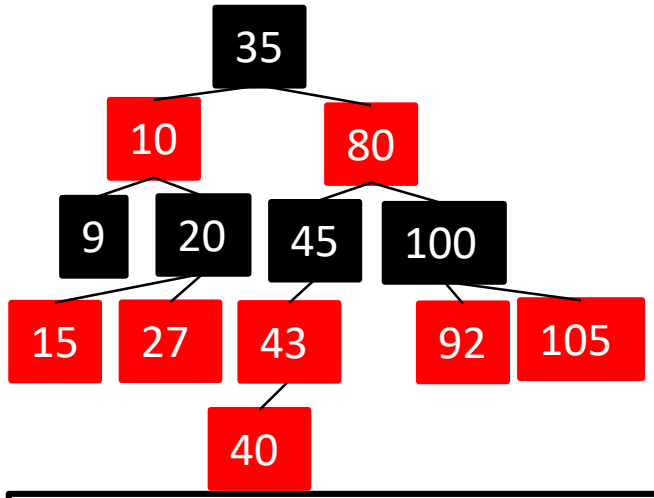
3. Pare negre ja està

Inserció: Red Black Tree. Cas Esquerre-Dret

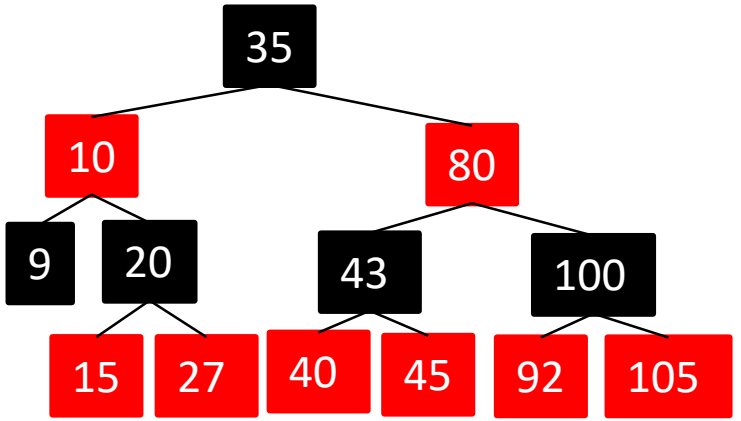
Inserció: 43



1. Cerca binària.
2. Inserim vermell

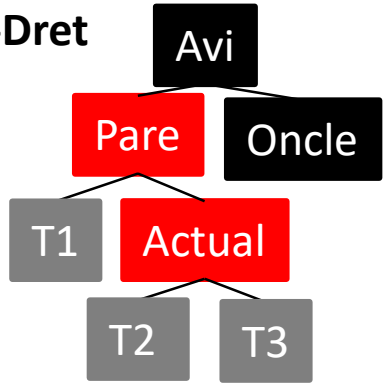


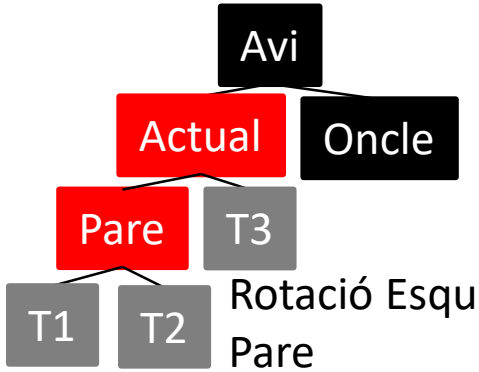
3. Pare vermell, oncle negre o null
4. Cas Esquerre-Dret
4-1. Rotació Esquerra pare



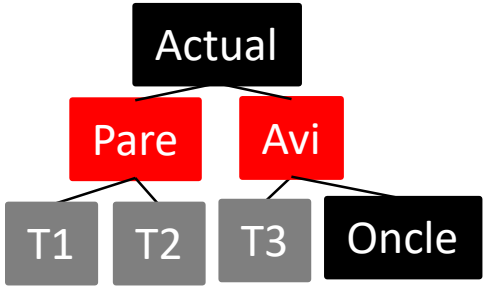
4.2 cas esquerre-esquerre
4.2.1 canvi color 43-45
43.2. rotació dreta 45

Cas Esquerre-Dret





Rotació Esquerre Pare

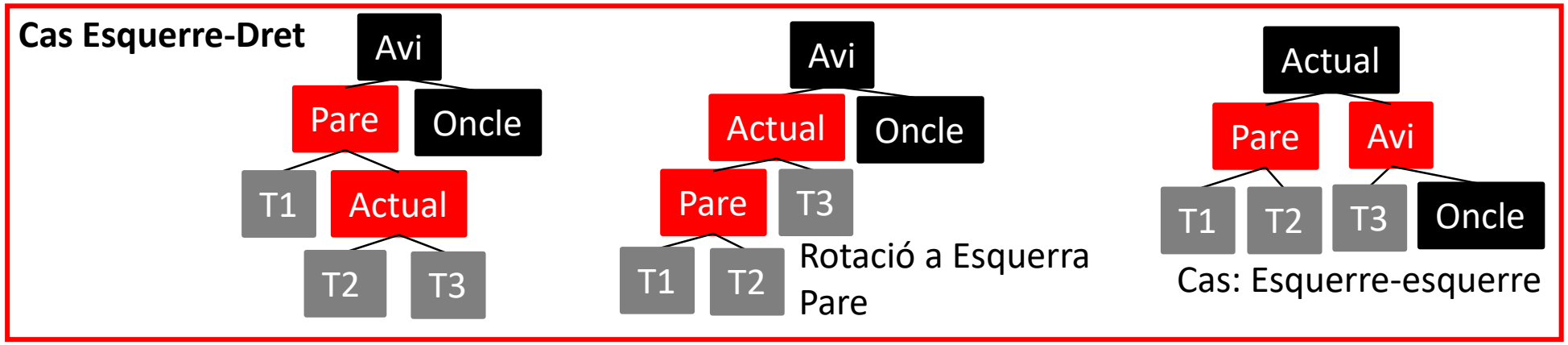


Cas: Esquerre-esquerre

Exercici

Modifica mètode rota pel cas esquerre-dret:

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



Conclusions Arbres equilibrats: AVL vs Red-Black

Arbres AVL (Adelson-**V**elskii i **L**andis, 1962)**:** Per a cada node la diferència d'alçada entre els arbres esquerre i dret no pot ser superior a 1.

- La diferència d'alçades entre dos subarbres germans és menor a 2.
- S´ha de re-equilibrar l'arbre si la diferència entre 2 germans és major a 1.
- Cercar un element és molt ràpid doncs estan estrictament equilibrats.

Arbres Vermell-Negre(Red-Black): Els nodes es classifiquen com a Vermells o Negres de la següent manera: L'arrel és negra. Els fills d'un node vermell són negres. Tot camí de l'arrel a una fulla passa pel mateix nombre de nodes negres.

Comparació:

- Cercar un element a AVL és més ràpid doncs estan estrictament equilibrats.
- Inserir i esborrar elements és més ràpid a Red-Black perquè es fan menys rotacions donat que el reequilibrat és més relaxat.
- Els arbres AVL necessiten **O(N)** de memòria extra per guardar les alçades de cada node. Mentre que Red-Black només necessita **O(1**) d'espai extra doncs amb un bit d'informació per node en té prou.
- La majoria de llibreries fan servir Red-Black: map, multimap i multiset a stl.