

Sessió 19 Arbres

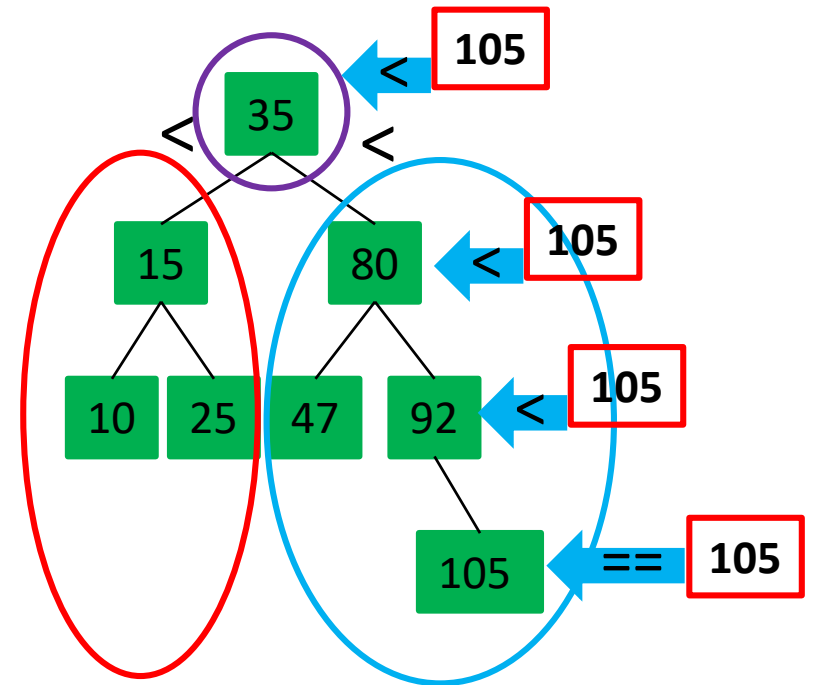
LP 2019-20

Recordem: Arbre de cerca binari

Arbre de cerca binari: Arbre binari ordenat on tot node és **major** que els nodes del seu subarbre **esquerre** i **menor** que els nodes del seu subarbre **dret**.

- Si busquem un valor mirem:
 - =arrel: **trobat**
 - <arrel: **busquem subarbre esquerre**
 - >arrel: **busquem subarbre dret**

Exemple: Busquem 105:
Quants nodes he visitat? 4
Quants nodes té l'arbre? 8

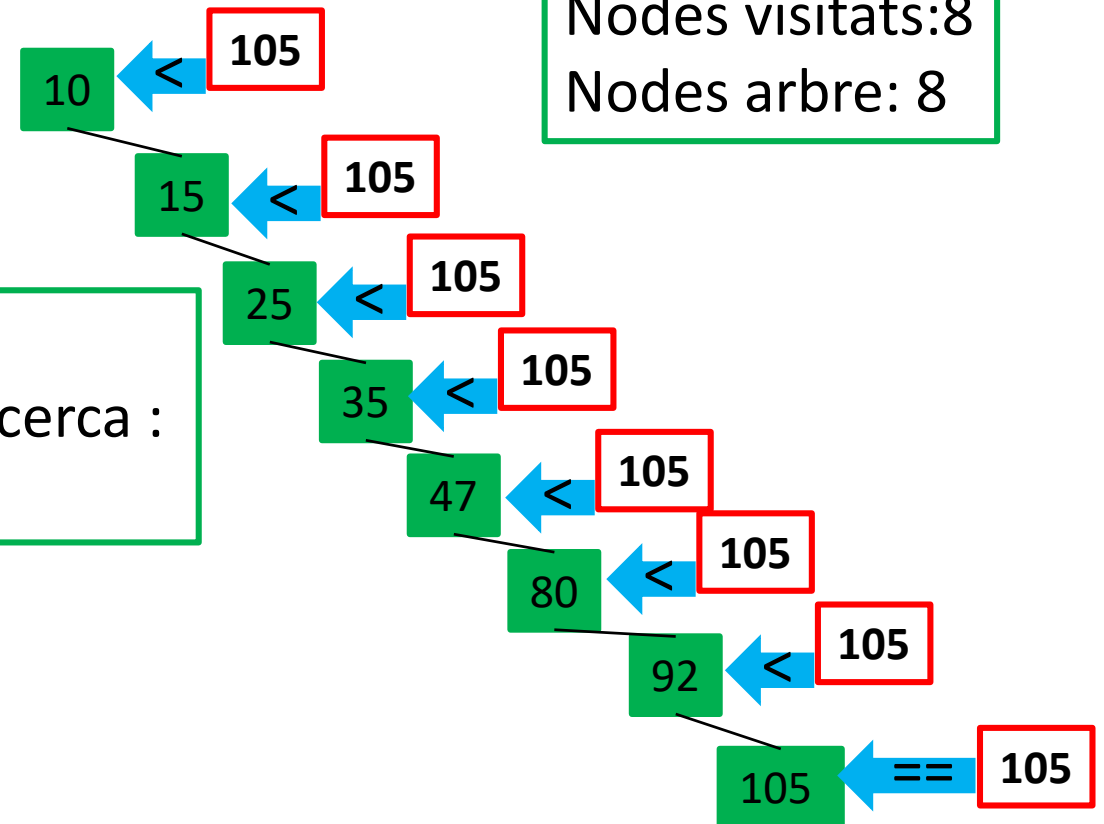


Arbre de cerca binari

Què passaria si el nostre arbre fos com aquest?

Exemple:

Busquem 105:
Nodes visitats: 8
Nodes arbre: 8



- a) Podem afegir i treure valors com vulguem.
- b) Si l'arbre està ordenat i és complet cost de cerca :
 $O(\log n)$ si no $O(n)$

Per aconseguir-ho alçada de l'arbre ha de ser $\log n$

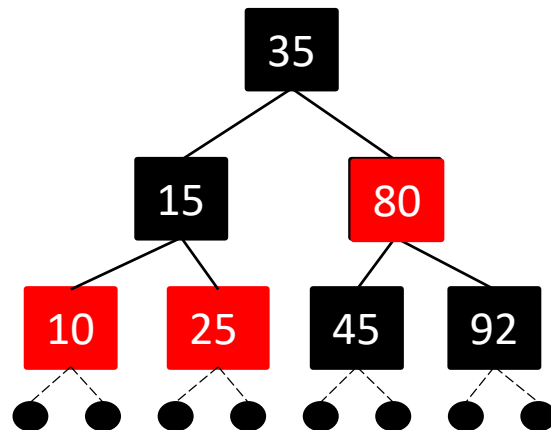
Arbre equilibrat

- **Arbre equilibrat:** És un arbre binari de cerca amb unes restriccions que li permeten garantir una alçada $\log n$. Per aconseguir-ho les operacions d'inserció i esborrat tenen més etapes.
- **Alguns tipus:**
 - **Arbres AVL (Adelson-Velskii i Landis, 1962 "An algorithm for the organization of information."):** Per a cada node la diferència d'alçada entre els arbres esquerre i dret no pot ser superior a 1.
 - **Arbres Vermell-Negre(Red-Black):** Els nodes es classifiquen com a Vermells o Negres segons:
 - 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.
 - **Splay-Trees:** Cada cop que s'accedeix a un node s'eleva a l'arbre passant a ser l'arrel (equilibrat promig).
 - **B-tree** (Bayer, McCreight) i **B+-tree** : arbres equilibrats usats per indexar els continguts de les bases de dades. En general, grau de l'arbre > 2 .

Red Black Tree

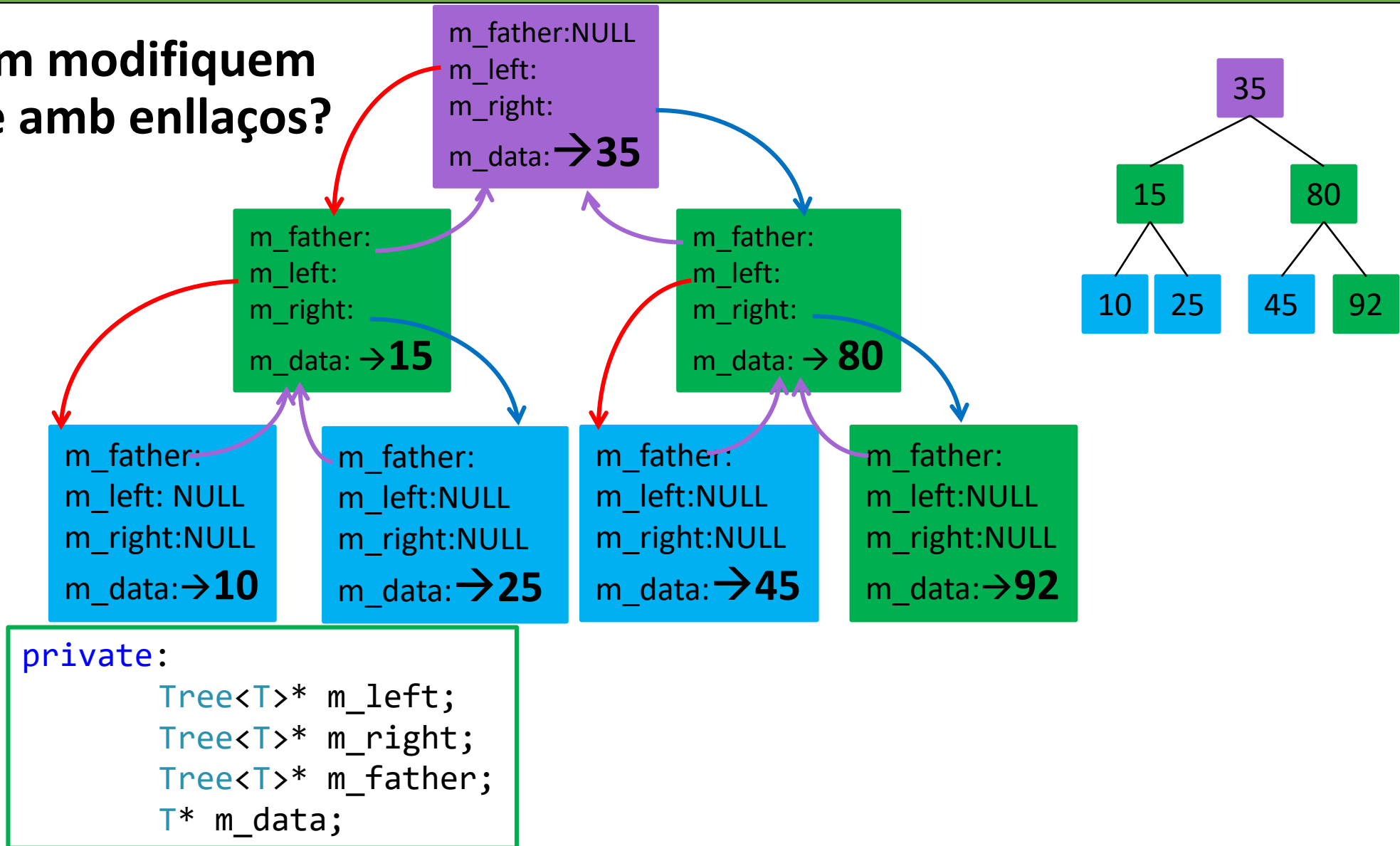
Un arbre Vermell- Negre ha de complir:

- Cada node té un color: vermell (R)/ o negre (B).
- L'arrel és negra.
- No hi pot haver dos nodes vermells consecutius a un camí. Per això un node vermell ha de tenir el pare i els fills negres.
- Qualsevol camí de l'arrel a una fulla té el mateix nombre de nodes negres. Considerem que els fills NULLS d'un node són nodes negres.
- Quan afegim un nou node inicialment és vermell.



Atributs Arbres Binari

1. Com modifiquem Arbres amb enllaços?



Red Black Tree: Implementació

```
template <class T> class TreeRB
{ public:
    TreeRB();
    ~TreeRB();
    bool isLeave();
    bool isEmpty() const ;
    bool cerca(const T& val, TreeRB<T>* valTrobat);
    TreeRB<T>* oncle();
    bool esFillDret();
    bool esFillEsq();
    friend std::ostream& operator<<(std::ostream& out, const TreeRB<T>& t)
    void insert(T& val);
private:
    enum COLOR { RED, BLACK };
    TreeRB<T>* m_left;
    TreeRB<T>* m_right;
    TreeRB<T>* m_father;
    T* m_data;
    COLOR m_color;
    void TreeRBRec(ifstream& fitxerNodes, int h, TreeRB<T>* father);
    std::ostream& coutArbreRec(int n, std::ostream& out) const;
};
```

Afegim m_color als nodes
Modifiquem constructors i operator<<
Per tenir en compte aquest nou atribut

Red Black Tree

Quan es pot desbalancejar un arbre Red-Black?

- **Quan inserim nodes**
- **Quan esborrem nodes**

Com rebalancegem?

- 1. Recolorejant**
- 2. Rotant**

Veiem-ho accedint a <https://tommikaikkonen.github.io/rbtree/#> i seleccionant el cas 1

Red Black Tree

➤ Insereix el node 35

Observació: Un nou node sempre és vermell



1. Un node nou (**x**) és **RED**.
2. Si (**x**) és l'arrel el posem a **BLACK**

➤ Insereix el node 15

➤ Insereix el node 80

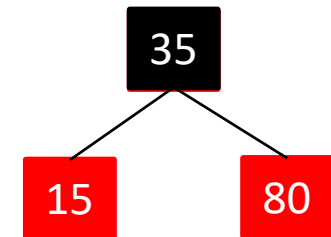
Observació: Per saber on inserir el node podem utilitzar la cerca binària

Red Black Tree. Inserció: RECOLOREJAT

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

Modifiquem cerca perquè ens retorni el node on es troba el valor o últim node visitat per tal d'inserir el nou node a partir d'ell.

Inserció:35
Inserció:15
Inserció:80




Red Black Tree: Modificació cerca

Modifiquem cerca perquè ens retorni node on es troba valor o últim node visitat

```
template<class T> bool TreeRB<T>::cerca(const T& val, TreeRB<T>*& valTrobat)
{
    if (m_data != NULL)
    {
        if (val == (*m_data)){valTrobat = this; return true; }
        else
        if (val < (*m_data))
        {
            if (m_left != NULL)
            {
                return m_left->cerca(val, valTrobat);
            }
            else valTrobat = this;
        }
        else
        {
            if (m_right != NULL)
            {
                return (m_right->cerca(val, valTrobat));
            }
            else valTrobat = this;
        }
    }
    return false;
}
```

Red Black Tree: Implementació

```
template <class T> class TreeRB
{ public:
    TreeRB() { m_left = NULL;
               m_right = NULL;
               m_father = NULL;
               m_data = NULL;
               m_color = RED;
    };
    void insert(T& val);
    //.....
private:
    enum COLOR { RED, BLACK };
    TreeRB<T>* m_left;
    TreeRB<T>* m_right;
    TreeRB<T>* m_father;
    T* m_data;
    COLOR m_color;
    void TreeBRec(ifstream& fitxerNodes, int h, TreeRB<T>* father);
    std::ostream& coutArbreRec(int n, std::ostream& out) const;
};
```



Quan inserim un nou node és **RED**

Red Black Tree: Implementació

```
template<class T> void TreeRB<T>::insert(T& val)
```

```
{  if (m_data == NULL)
    { //Arbre buit
      m_data = new T;
      (*m_data) = val;
      m_color = BLACK;
    }
}
```

Si (x) és l'arrel el posem a **BLACK**

Si no, el creem com a fill dret o esquerre segons la cerca

```
else
```

```
{  TreeRB<T>* ptAux=nullptr;
    bool trobat = cerca(val, ptAux);
    if (!trobat)
    { //Si no trobem valor creem node RED amb val fill de ptAux
      TreeRB<T>* nouNode = new TreeRB<T>;
      nouNode->m_data = new T;
      *(nouNode->m_data) = val;
      nouNode->m_father = ptAux;
      if (val < *(ptAux->m_data))
      else
    }
}
```

```
//El nou valor sera fillEsq de ptAux
{ptAux->m_left= nouNode; }
{ptAux->m_right= nouNode;}
```

```
//El nou valor sera fillDret de ptAux
```

Red Black Tree. Inserció: RECOLOREJAT

Tornem a l'aplicació:

➤ Insereix el node 92

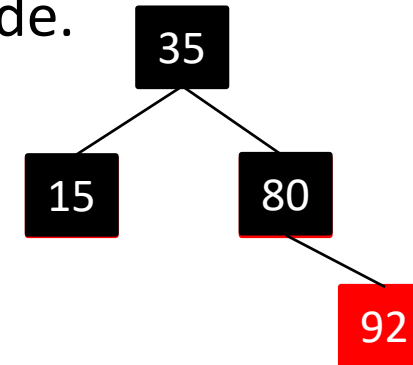
Quan tenim dos nodes **RED** seguits hem d'arreglar arbre

Què podem fer?

Red Black Tree. Inserció: RECOLOREJAT

Inserció:92

- ➡ 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**)
 - ➡ i. Canvia color de pare i oncle a **BLACK**



Quan tenim dos nodes **RED** seguits hem d'arreglar arbre

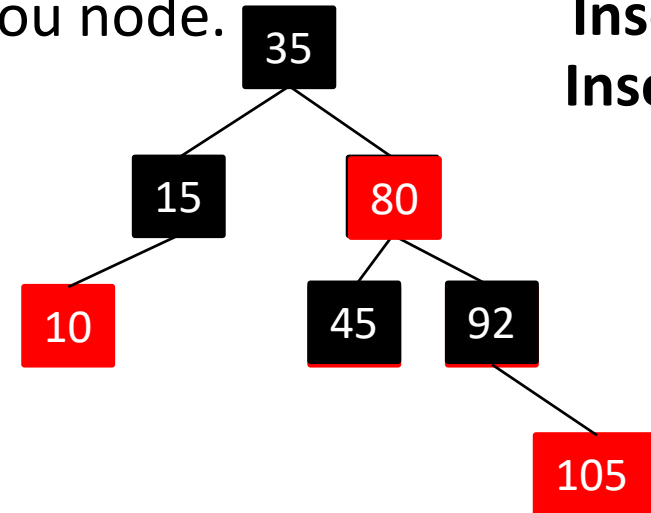
Red Black Tree. Inserció: RECOLOREJAT

- Insereix els nodes 10, 45 i 105

Què ha passat?

Red Black Tree. Inserció: RECOLOREJAT

- ➡ 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**)
 - ➡ i. Canvia color de pare i oncle a **BLACK**
 - ➡ ii. Canvia color avi a **RED**



Inserció:10
Inserció:45
Inserció:105

Red Black Tree. Inserció: RECOLOREJAT

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

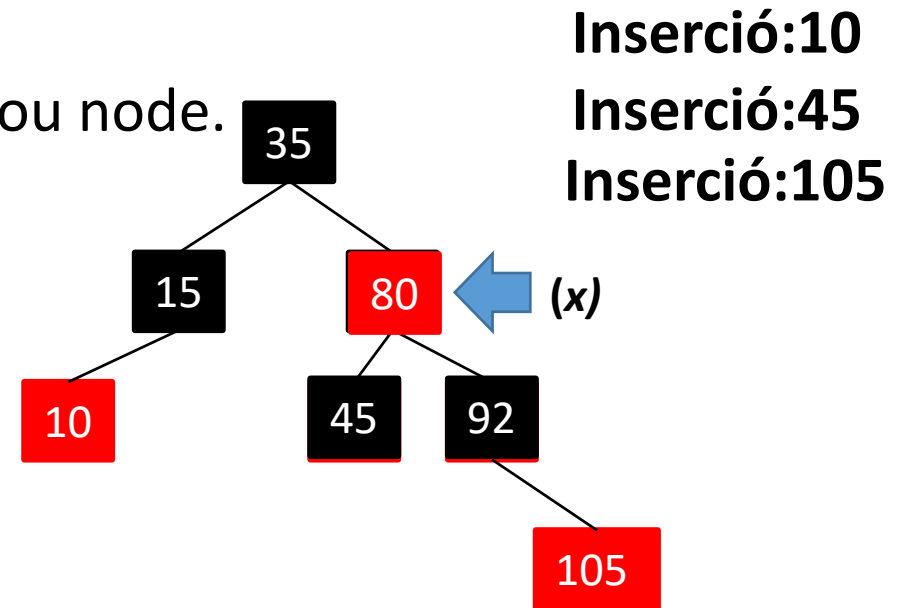
2. Sino (el pare és **RED**)

1. Si l'oncle és **RED** (avi ha de ser **BLACK**)

i. Canvia color de pare i oncle a **BLACK**

ii. Canvia color avi a **RED**

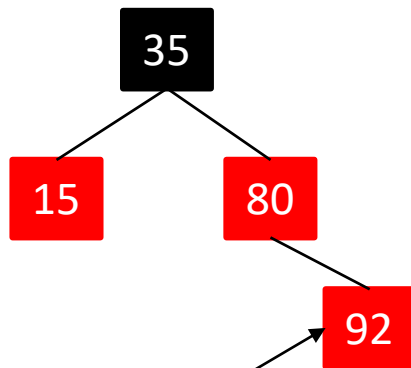
➔ iii. Canvia (**x**) *punt 2.* per avi i repeteix procés 3.



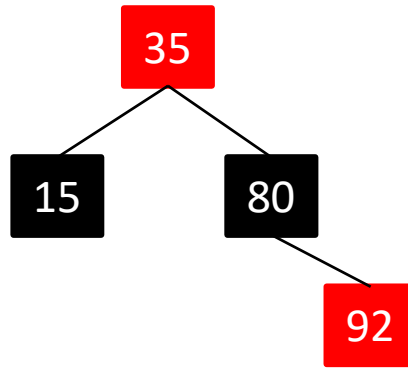
Ens hem d'assegurar que tot és correcte. Per exemple, ens podria haver quedat l'arrel vermella.

Recapitulem

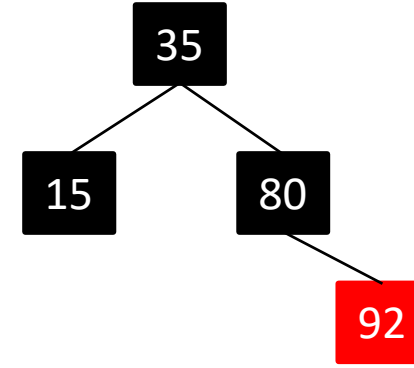
1. Inserció:92



1. cerca binària
Inserim vermell



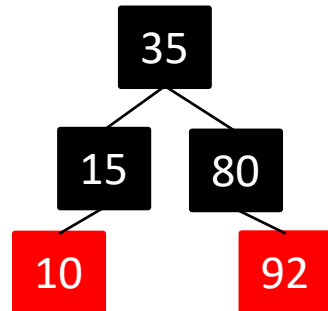
2. Pare i oncle vermells →
els posem negres
Avi negre el posem vermell



3. Mirem de forma recursiva avi
Si és arrel el posem a negre

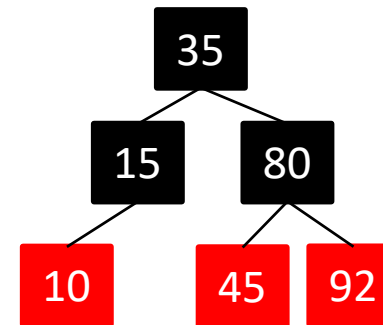
Recapitulem

1. Inserció:10



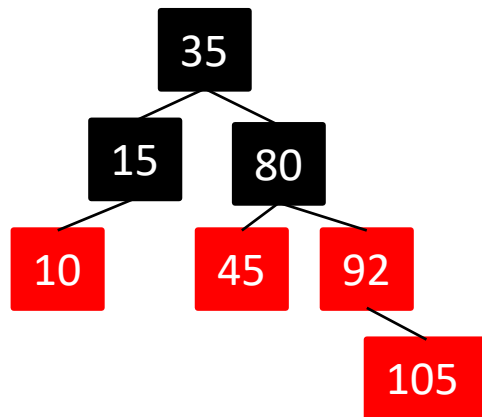
1. cerca binària.
2. Inserim vermell
3. Si pare és negre ja està

2. Inserció:45

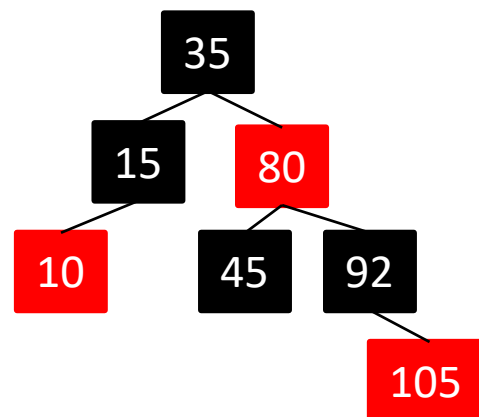


1. cerca binària.
2. Inserim vermell
3. Si pare és negre ja està

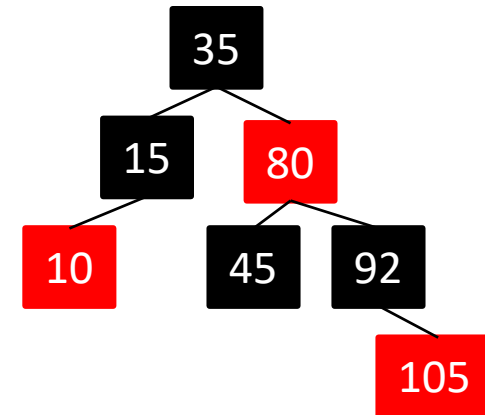
3. Inserció:105



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: Implementació

```
template<class T> void TreeRB<T>::insert(T& val)
```

```
{ if (m_data == NULL)
```

```
{ //Arbre buit
```

```
  m_data = new T;
```

```
  (*m_data) = val;
```

```
  m_color = BLACK;
```

```
}
```

```
else
```

```
{ TreeRB<T>* ptAux=nullptr;
```

```
  bool trobat = cerca(val, ptAux);
```

```
  if (!trobat)
```

```
  { //Si trobem el valor no fem res, si no creem node RED amb val fill de ptAux
```

```
    TreeRB<T>* nouNode = new TreeRB<T>;
```

```
    nouNode->m_data = new T;
```

```
    (*nouNode->m_data) = val;
```

```
    nouNode->m_father = ptAux;
```

```
    if (val < (*ptAux->m_data))
```

```
      else
```

```
  }
```

```
  arreglaREDRED(nouNode);
```

```
}
```

← Controlem si l'arbre és buit

Cerquem la posició on inserir el node

//El nou valor sera fillEsq de ptAux

{ptAux->m_left= nouNode; }

{ptAux->m_right= nouNode;}

//El nou valor sera fillDret de ptAux

Exercici

Implementa mètode:

```
template<class T>
```

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

nouNode és (x)
Mètode recursiu

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**)

- i. Canvia color de pare i tiet a **BLACK**

- ii. Canvia color avi a **RED**

- iii. Canvia (**x**) punt 2. per avi i repeteix procés 3.

Necessitem implementar
mètode oncle

Exercici

Implementa mètodes:

```
template<class T>  
TreeRB<T>* TreeRB<T>::oncle();
```

```
template<class T>  
bool TreeRB<T>::esFillDret();
```

```
template<class T>  
bool TreeRB<T>::esFillEsq();
```

Exercici

Implementa mètode:

```
template<class T>
```

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

nouNode és (x)
Mètode recursiu

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**)

i. Canvia color de pare i tiet a **BLACK**

ii. Canvia color avi a **RED**

iii. Canvia (**x**) *punt 2.* per avi i repeteix procés 3.