

Sessió 14. Exercicis heaps i d'Arbres

Estructura de Dades
Curs 2020-2021

Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona

Contingut

Problema 1 – Construir Min Heap

Problema 2 – Construir Max Heap i removeMax

Problema 3 – Construir Arbre de cerca binària

Problema 4 – Funció recursiva mirall

Problema 5 – Funció recursiva sumaMaximaAFulles

Problema 6 – Funció recursiva mostraAncestreComuMesBaix

Problema 1

A partir d'un **min-heap** inicialment buit. Inseriu els elements següents en l'ordre especificat:

60, 70, 90, 35, 50, 65, 21, 63, 64, 61

Feu el dibuix de la inserció de tots els elements i indiqueu quins nodes han fet upheap.

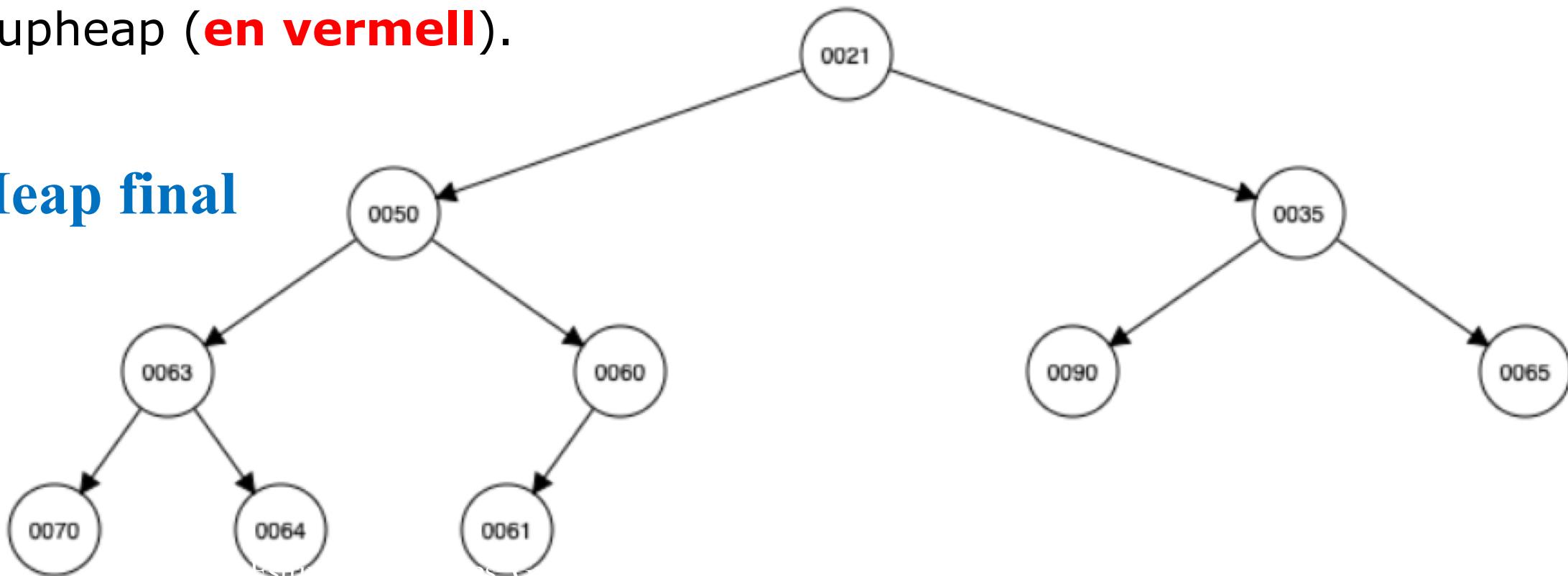
Problema 1 (SOLUCIÓ)

A partir d'un min-heap inicialment buit. Inseriu els elements següents en l'ordre especificat:

60, 70, 90, 35, 50, 65, 21, 63, 64, 61

Feu el dibuix de la inserció de tots els elements i indiqueu quins nodes han fet upheap (**en vermell**).

Heap final



Problema 2 (PART I)

A partir d'un **max-heap** inicialment buit. Inseriu els elements següents en l'ordre especificat:

60, 70, 90, 35, 50, 65, 21, 63, 64, 61

Feu el dibuix de la inserció de tots els elements i indiqueu quins nodes han fet upheap.

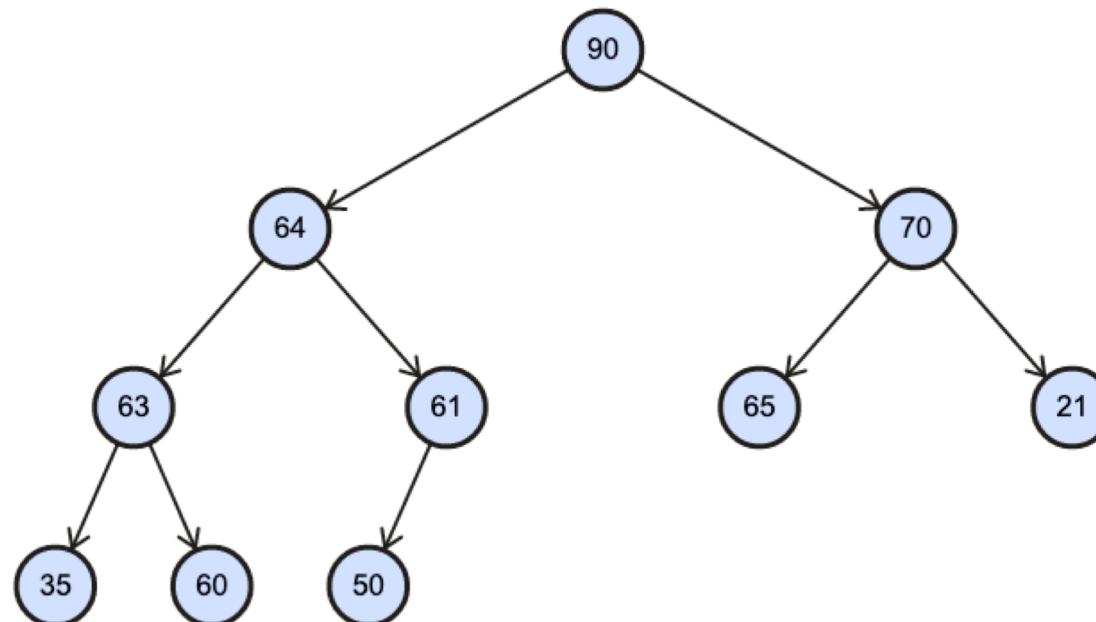
Problema 2 (PART I - Solució)

A partir d'un **max-heap** inicialment buit. Inseriu els elements següents en l'ordre especificat:

60, 70, 90, 35, 50, 65, 21, 63, 64, 61

Feu el dibuix de la inserció de tots els elements i indiqueu quins nodes han fet upheap (**en vermell**).

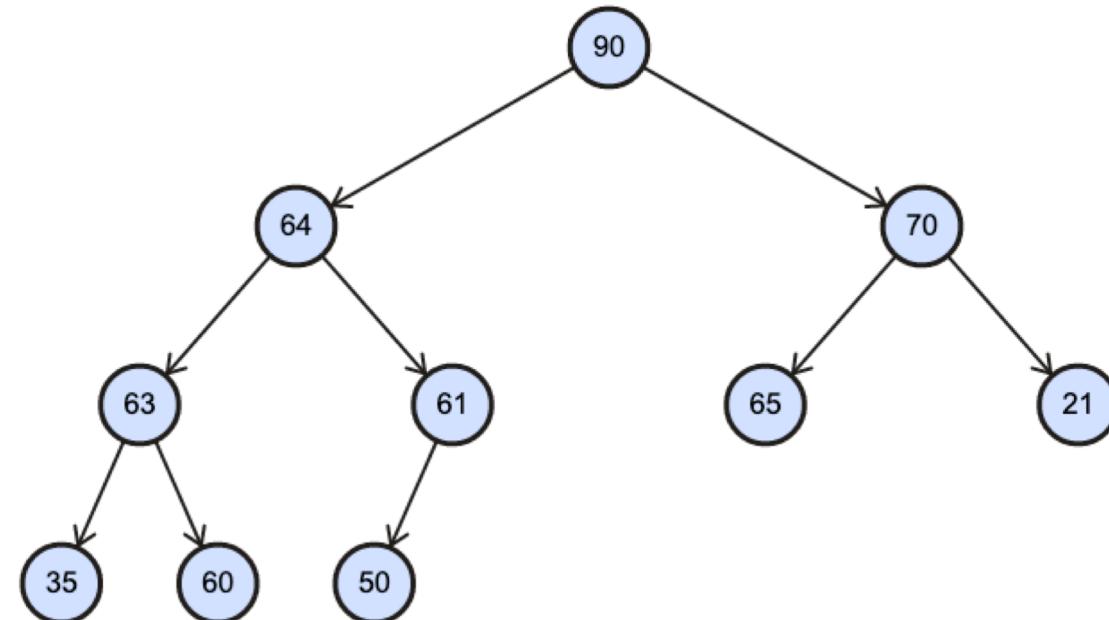
Heap final



Problema 2 (PART II)

A partir d'un max-heap construït. Feu 3 crides removeMax() i dibuixeu l'arbre en cada crida.

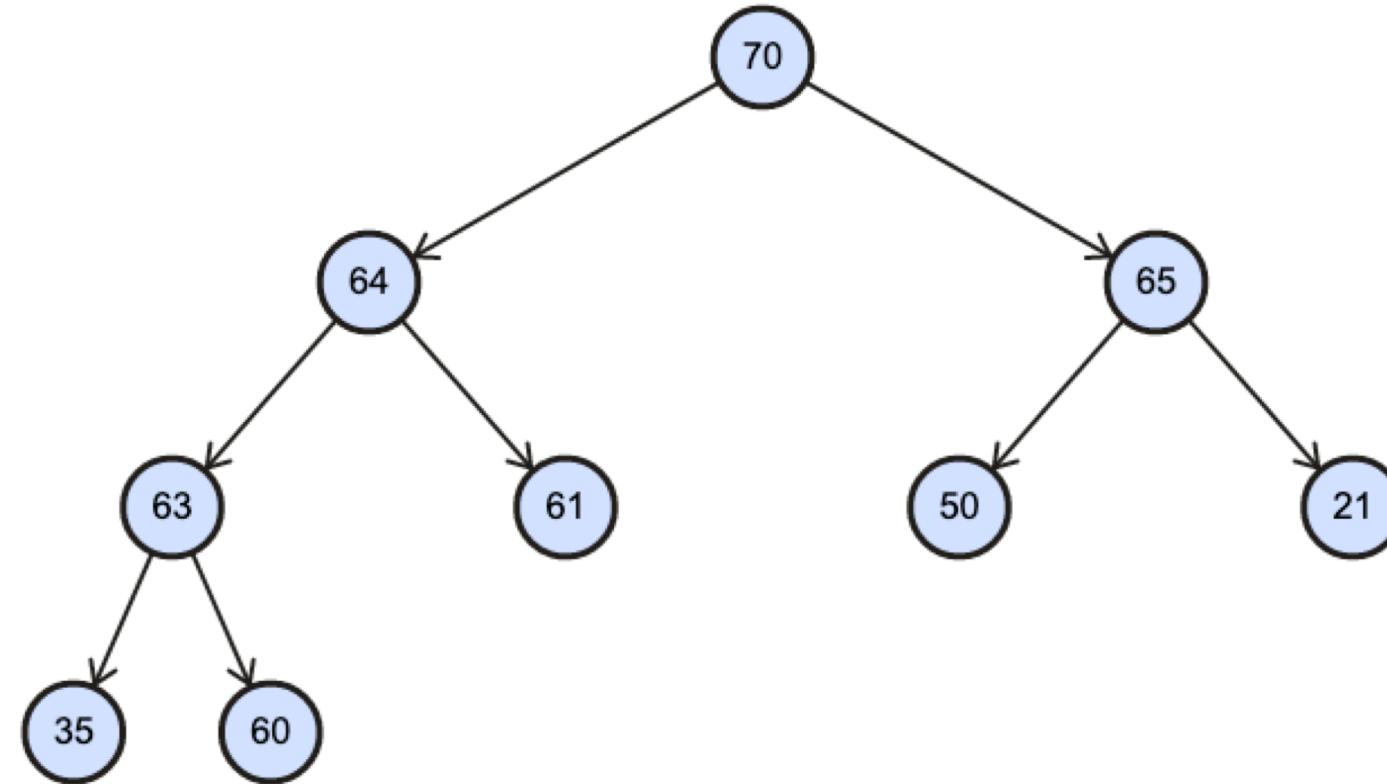
Heap Inicial



Problema 2 (PART II Solució)

A partir d'un max-heap construït. Feu 3 removeMax() i dibuixeu l'arbre en cada crida.

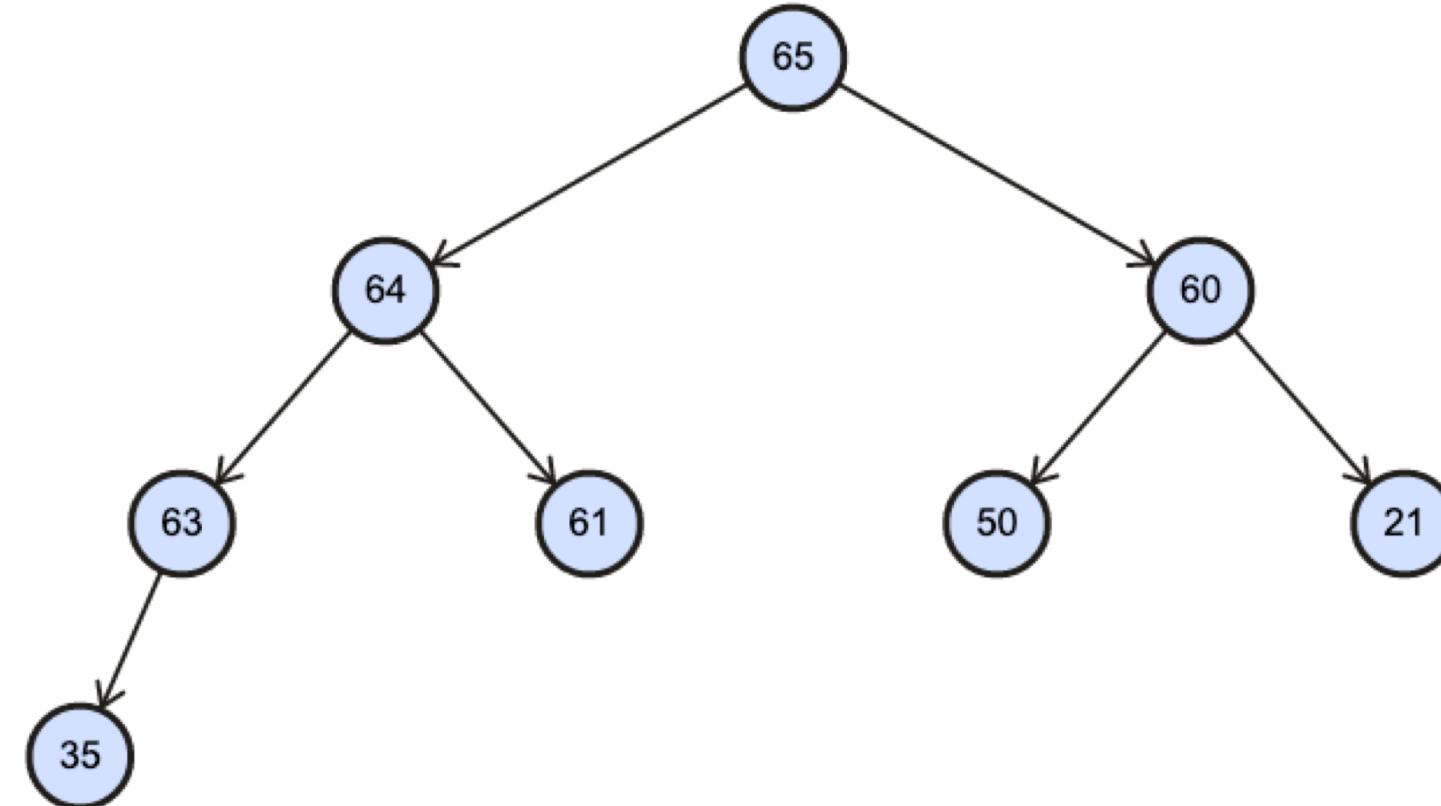
removeMax 1



Problema 2 (PART II Solució)

A partir d'un max-heap construït. Feu 3 removeMax() i dibuixeu l'arbre en cada crida.

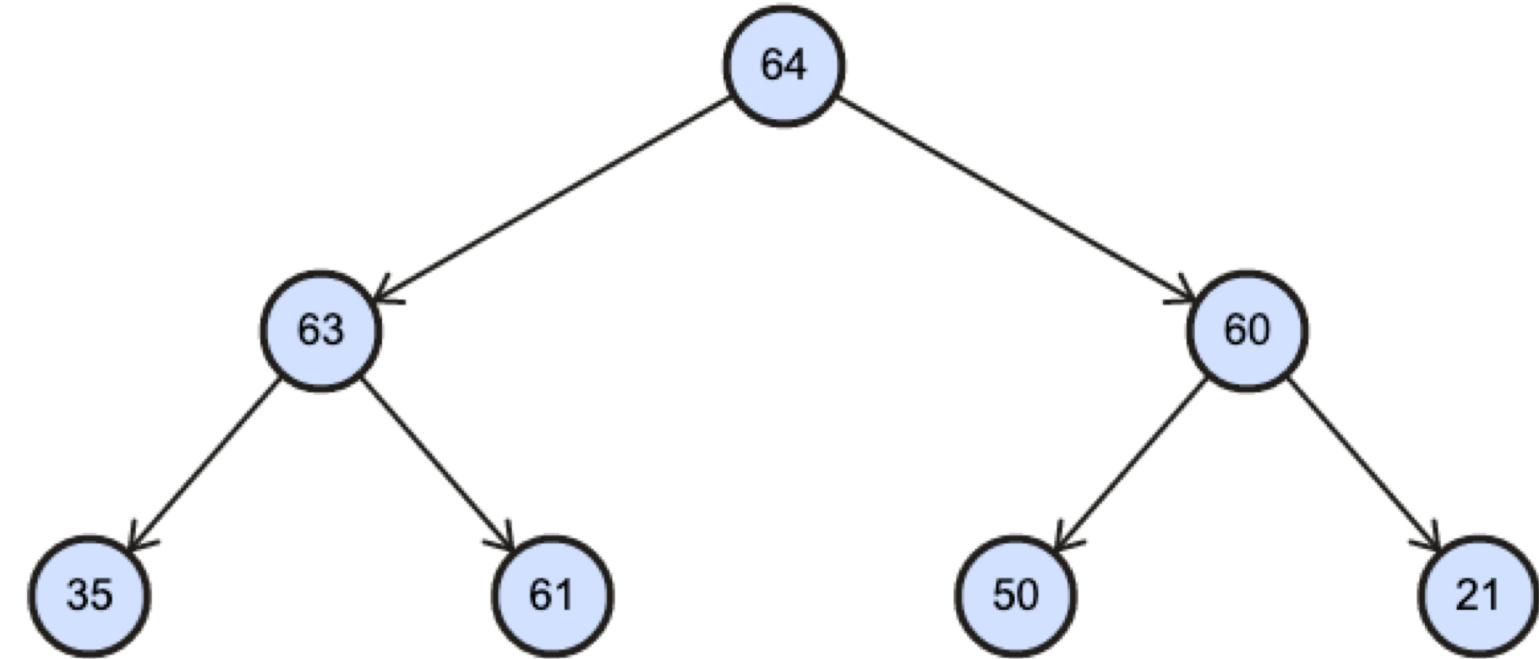
removeMax 2



Problema 2 (PART II Solució)

A partir d'un max-heap construït. Feu 3 removeMax() i dibuixeu l'arbre en cada crida.

removeMax 3



Problema 3

El recorregut en preordre d'un arbre de cerca binària és:

30, 20, 10, 15, 25, 23, 39, 35, 42

Dibuixa l'arbre de cerca binària que permet aquest recorregut en preordre i indica quin és el recorregut en postordre pel mateix arbre.

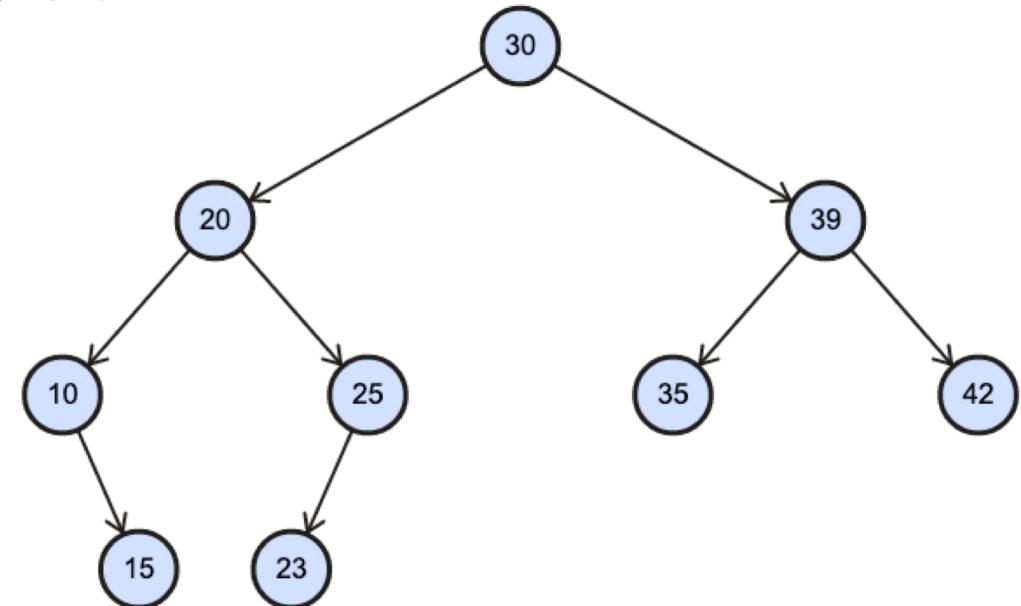
Problema 3 (SOLUCIO)

El recorregut en preordre d'un arbre de cerca binària és:

30, 20, 10, 15, 25, 23, 39, 35, 42

Dibuixa l'arbre de cerca binària que permet aquest recorregut en preordre i indica quin és el recorregut en postordre pel mateix arbre.

Postordre = 15, 10, 23, 25, 20, 35, 42, 39, 30



Problema 4

Implementeu recursivament un mètode anomenat `mirall()` de la classe `BSTree` (un arbre de cerca binària). Aquest mètode realitza l'operació mirall sobre l'arbre. Suposeu que l'arbre és d'enters i que el `BSTree` té definit una funció `root()` que retorna el `root_node` que és de tipus `NodeTree *`.

Els `NodeTree` tenen un atribut `_right`, un atribut `_left` i un atribut `_element` privats que s'accedeixen mitjançant les funcions `right()`, `left()`, `element()`, i també teniu una funció `isExternal()` per retornar si un node és fulla o no.

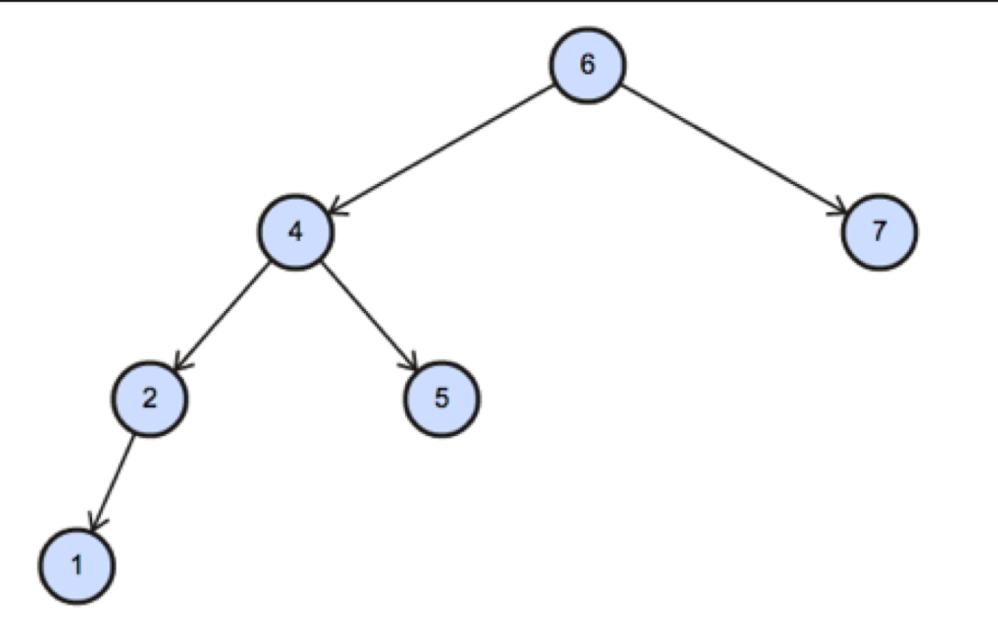
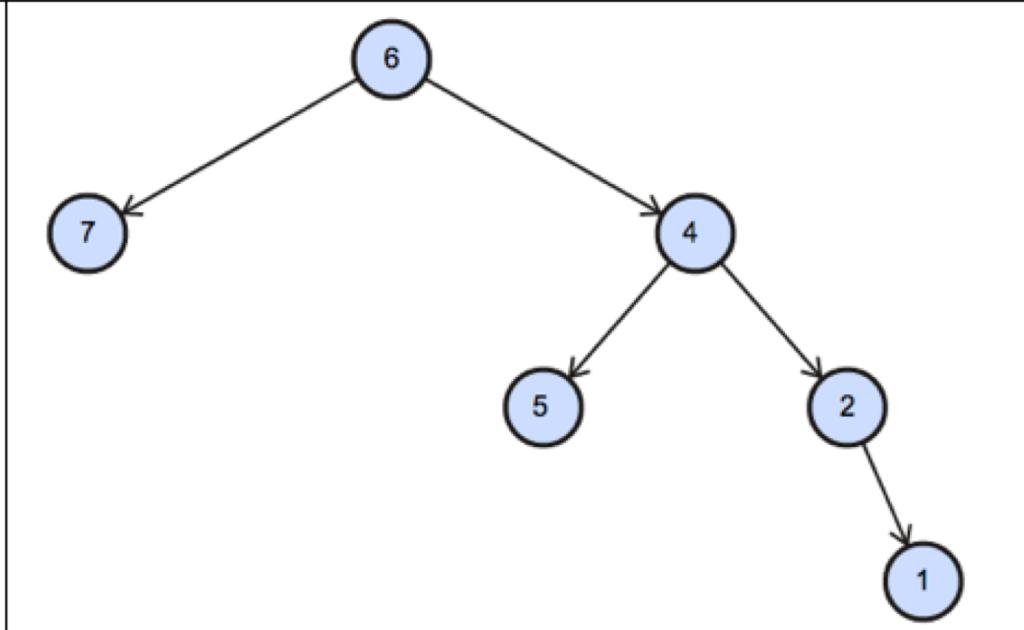
```
template <class Element>
void BSTree<Element>::mirall()
{
    Aquí el vostre codi
}
```

Definiu aquí les funcions auxiliars que necessiteu

Problema 4

Al següent exemple s'observa aquesta operació. A l'arbre de l'esquerra, quan es crida a l'operació mirall queda com es visualitza l'arbre de la dreta.

Mireu amb atenció la funció que heu d'implementar i noteu que l'arbre no es retorna. No s'ha de crear un arbre auxiliar ni utilitzar cap altre estructura de dades per a la solució, sinó que s'ha de modificar directament l'arbre intern del TAD de forma recursiva.

 <p>T = {6,4,2,1,5,7} (preordre)</p>	 <p>T={6,7,4,5,2,1} (preordre, després d'aplicar mirall)</p>
-------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------

Problema 4 (Solució)

```
template <class Element> void BSTree<Element>::mirall()
{
    this->mirrorTree(this->root());
}

template <class Element> void BSTree<Element>::mirrorTree(NodeTree<Element> * node)
{
    if (!node->isExternal()) {
        if (node->left() != nullptr) this->mirrorTree(node->left());
        if (node->right() != nullptr) this->mirrorTree(node->right());

        NodeTree<Element> * esq = node->left();
        NodeTree<Element> * dre = node->right();

        node->setLeft(dre);
        node->setRight(esq);

    } else return;
}
```

Solució no
optimitzada !!

Problema 5

Implementeu recursivament un mètode anomenat `sumaMaximaAFulles()` de la classe `BSTree` (un arbre de cerca binària). Aquest mètode calcula la suma màxima de camins possibles des de les fulles a l'arrel. Supposeu que l'arbre és d'enters i que el `BSTree` té definit una funció `root()` que retorna el `root_node` que és de tipus `NodeTree *`.

Els `NodeTree` tenen un atribut `_right`, un atribut `_left` i un atribut `_element` privats que s'accedeixen mitjançant les funcions `right()`, `left()`, `element()`, i també teniu una funció `isExternal()` per retornar si un node és fulla o no.

```
template <class Element>

int BSTree<Element>::sumaMaximaAFulles()
{
    Aquí el vostre codi
}
```

Definiu aquí les funcions auxiliars que necessiteu

Problema 5

- **La suma màxima dels camins possibles a la fulla des de l'arrel.**
- Donat un arbre de cerca binari T es demana la suma màxima dels camins possibles des de l'arrel a les fulles. En el cas que l'arbre estigui buit, la suma màxima de camins és 0.

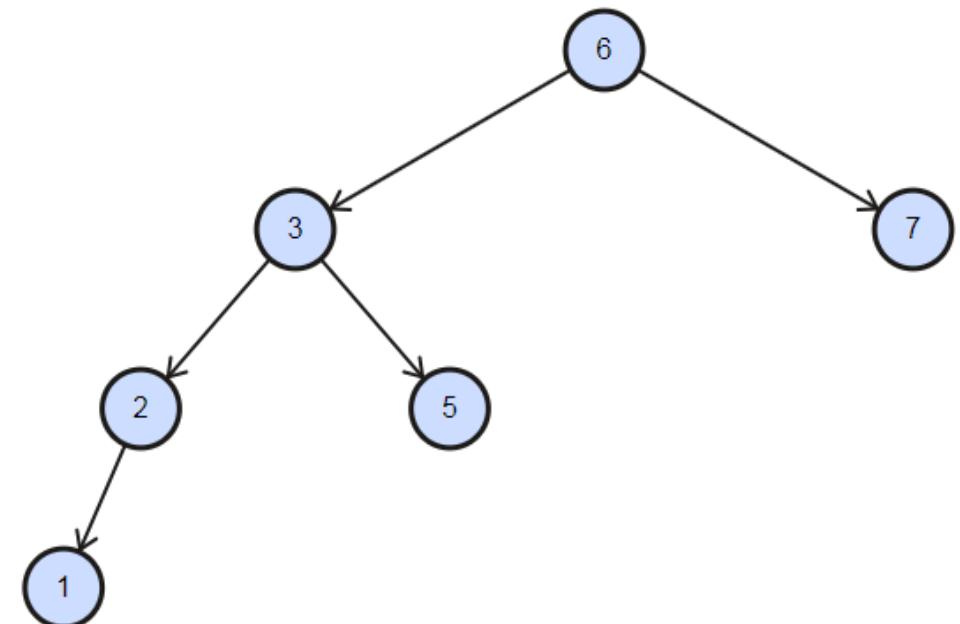
A l'exemple: els camins possibles són:

$$6+3+2+1 = 12$$

$$\mathbf{6+3+5 = 14}$$

$$6+7= 13$$

Per tant, la suma màxima de tots els camins possibles donarà com a resultat 14.



Problema 5 (Solució)

```
template <class Element>

int BSTree<Element>::sumaMaximaAFulles() {
    return this->sumaMaximaAFulles(this->root_node);
}

template <class Element>
int BSTree<Element>::sumaMaximaAFulles(NodeTree<Element>* root) {
    if (root == nullptr) return 0;
    int left = sumaMaximaAFulles(root->left());
    int right = sumaMaximaAFulles(root->right());
    return (left > right ? left : right) + root->getElement();
}
```

Solució no
optimitzada !!

Problema 6

Implementeu recursivament un mètode anomenat `mostraAncestreComuMesBaix()` de la classe `BSTree` (un arbre de cerca binària). Aquest mètode calcula l'ancestre comú a dos nombres enters que existeixen a l'arbre. Supposeu que l'arbre és d'enters i que el BSTree té definit una funció `root()` que retorna el `root_node` que és de tipus `NodeTree *`.

Els NodeTree tenen un atribut `_right`, un atribut `_left` i un atribut `_element` privats que s'accedeixen mitjançant les funcions `right()`, `left()`, `element()`, i també teniu una funció `isExternal()` per retornar si un node és fulla o no.

```
template <class Element>  
void BSTree<Element>::mostraAncestreComuMesBaix()  
{    Aquí el vostre codi  
}
```

Definiu aquí les funcions auxiliars que necessiteu

Problema 6

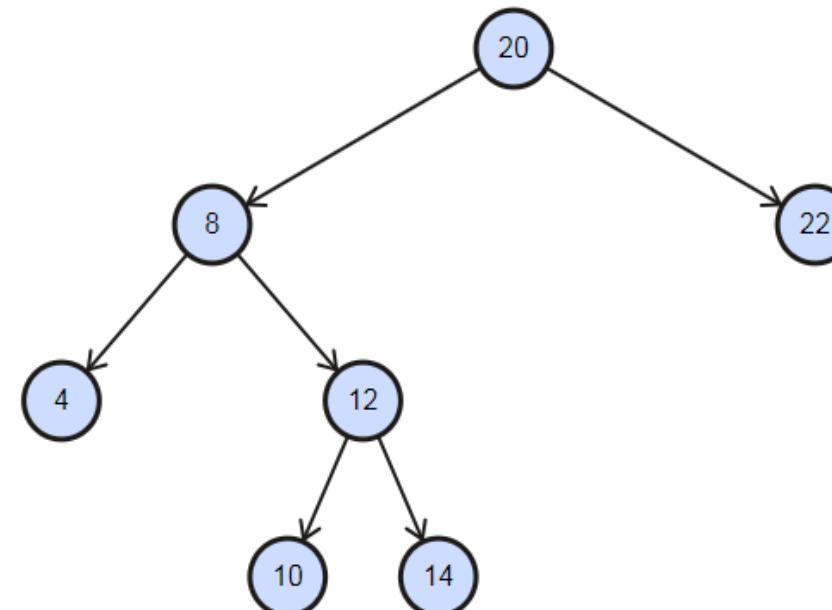
- **Mostra l'ancestre comú més baix (ACMB) en un arbre binari de cerca.**
- Donat un arbre T, definim l'ACMB entre dos nodes n1 i n2 com el node més baix en T que té n1 i n2 com a descendents (on permetem que un node sigui un descendant de si mateix), per tant l'ACMB de n1 i n2 en T és l'antecessor compartit de n1 i n2 que es troba més allunyat de l'arrel.
- Important, considerem que els valors n1 i n2 sempre es troben a l'arbre. I que l'arbre no té elements repetits.

A l'exemple:

L'ACMB de 10 i 14 és: 12

L'ACMB de 14 i 8 és: 8

L'ACMB de 10 i 22 és: 20



Problema 6 (Solució)

```
template <class Element>
void BSTree<Element>::mostraAncestreComuMesBaix(const Element& n1, const Element& n2)
{
    cout << "L'ACMB de " << n1 << " i " << n2 << " es: "
        << this->acmb(this->root(),n1,n2) << endl;
}

template <class Element>
int BSTree<Element>::acmb(NodeTree<Element>* node, const Element& n1, const Element& n2)
{
    if (node == nullptr) return -1;
    if (node->element() > n1 && node->element() > n2)
        return acmb(node->left(), n1, n2);
    if (node->element() < n1 && node->element() < n2)
        return acmb(node->right(), n1, n2);

    return node->element();
}
```

Solució no
optimitzada !!