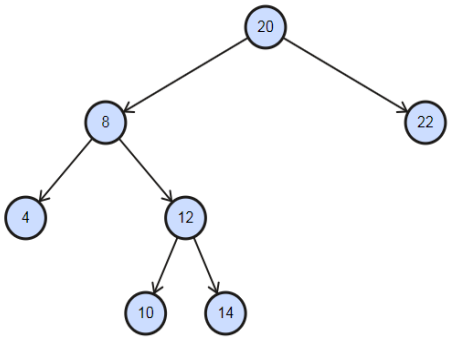
Nom i Cognoms: _____ Signatura: _____

Abans de començar:

- **CAMPUS VIRTUAL:**
  - Descarrega't el projecte que vas lliurar el diumenge dia 16/5/2021
  - Descarrega't el main del campus virtual.
- En aquesta prova s'ha d'utilitzar **Netbeans 8.2, C++ versió 11**
- S'ha de lliurar al Campus Virtual **un únic fitxer zip** amb el vostre nom i cognoms. Per exemple **LisaSimpson_control_P3.zip**. Aquest fitxer contindrà el codi de tots els exercicis del lliurament, incloent les funcions dels dos exercicis de la prova.
- ***Utilitzeu el projecte de l'exercici 1 com a projecte base per fer la prova***
  - Cada exercici de la prova és una nova funció a implementar en el TAD BinarySearchTree de l'exercici 1 de la pràctica.
  - Penseu que els exercicis estan explicats amb templates, en cas que hagueu implementat el TAD sense templates, elimineu els templates al vostre codi. En cas que tingueu la implementació amb templates, mantingueu la definició que se us indica.
  - En el vostre main.cpp, incorporeu les dues funcions per testejar cada exercici de la prova.

(3.0 p.) EXERCICI 1. Comprovar si tots els nodes interns de l'arbre tenen dos fills.
Donat un arbre T, volem saber si l'arbre T té tots els nodes interns amb dos fills.



Fixeu-vos en l'exemple.

El mètode *haveAllNodesTwoChildren()* retorna true perquè els nodes interns de l'arbre tots tenen 2 fills. En canvi, si al mateix arbre afegim un 21, 30 i 25 i el mètode *haveAllNodesTwoChildren()* retornarà false perquè hi ha nodes de l'arbre que no tenen 2 fills.

Es demana el següent:

- Agafeu la classe BST i implementeu els mètodes següents:

```
// Mètode que retorna cert si l'arbre té tots els nodes amb 2 fills.  False en cas contrari.
template <class K, class V>
bool BinarySearchTree<K,V>::haveAllNodesTwoChildren()
{
                    // codi a implementar
}

// Mètode que retorna TRUE si l'arbre té tots els nodes interns amb 2 fills;  FALSE  si l'arbre té nodes interns amb més d'un fill
// aquesta funció us ha de servir per fer la recursivitat
template <class K, class V>
bool BinarySearchTree<K,V>::haveAllNodesTwoChildren(BinaryTreeNode<K,V>* node) {
                    // codi a implementar
}
```

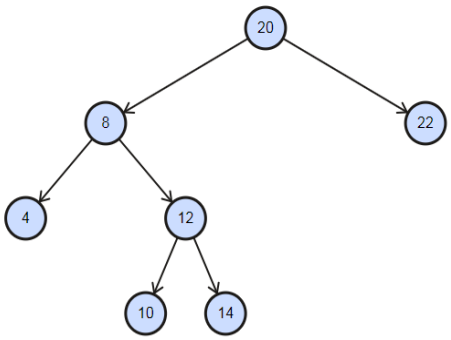
No podeu afegir més mètodes auxiliars i no podeu modificar el main o les funcions. Podeu eliminar els templates si no en teniu als vostres TADs.

(3.0 p.) EXERCICI 2. 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 descendent 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.**

Suposem un arbre binari de cerca, on cada node conté un enter únic (no pot haver dos nodes amb el mateix enter).

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



Es demana el següent:

- Agafeu la classe BinarySearchTree i implementeu els mètodes següents:

```
// Donats els valors de dos nodes n1 i n2 en un arbre binari de cerca, mostra per pantalla l'ancestre comú més baix (ACMB).  // Pots suposar que els dos valors existeixen a l'arbre i que són únics.
template <class K, class V>
void BinarySearchTree<K,V>::mostraAncestreComuMesBaix(const K& n1, const K& n2)
{
        // codi a implementar
}

// Donats els valors de dos nodes n1 i n2 en un arbre binari de cerca amb arrel root,  retorna l'ancestre comú més baix (ACMB).   Aquesta funció us ha de servir per fer la recursivitat
template <class K, class V>
int BinarySearchTree<K,V>::acmb(BinaryTreeNode<K,V>* node,  const K& n1, const K& n2)
{
                // codi a implementar
}
```

No podeu afegir més mètodes auxiliars i no podeu modificar el main o les funcions. Podeu eliminar els templates si no en teniu als vostres TADs.