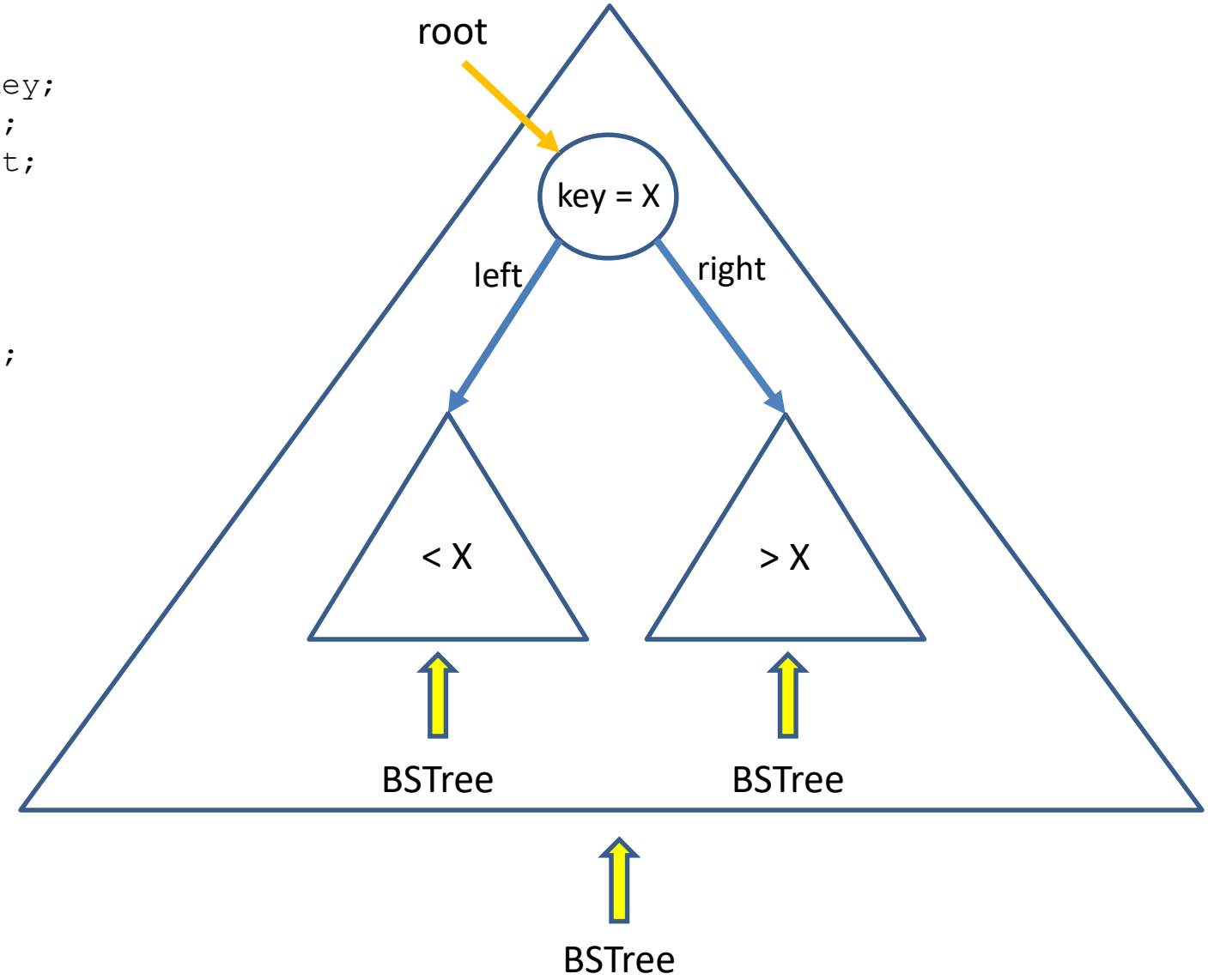
Algoritmi in podatkovne strukture 1

9. laboratorijske vaje
Binarna iskalna drevesa

BINARNO ISKALNO DREVO

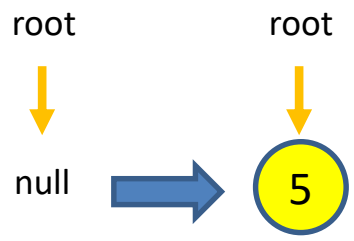
```
class BSTNode {
    Comparable key;
    BSTNode left;
    BSTNode right;
    …
}

class BSTree {
    BSTNode root;
    …
}
```

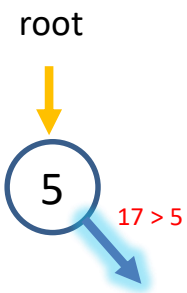


VSTAVLJANJE V BST

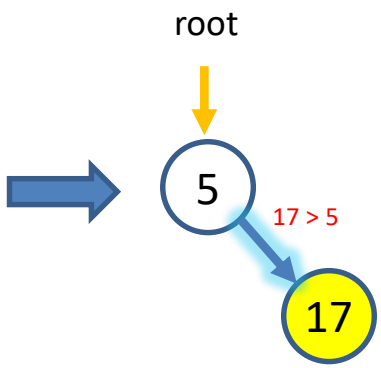
Na začetku je binarno iskalno drevo prazno. Nariši drevo po končanem vstavljanju elementov: 5, 17, 7, 2, 11, 9, 1, 6.

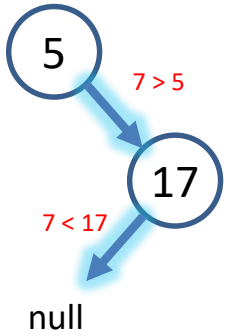


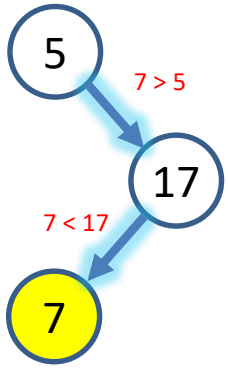
prvi element postane koren drevesa



prostor za vstavitev novega elementa

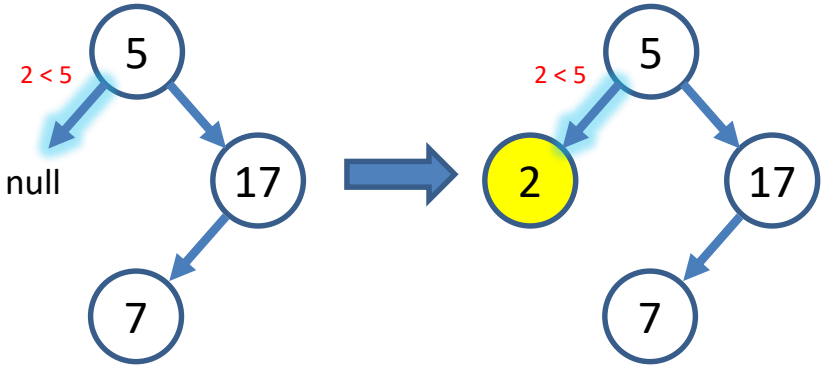


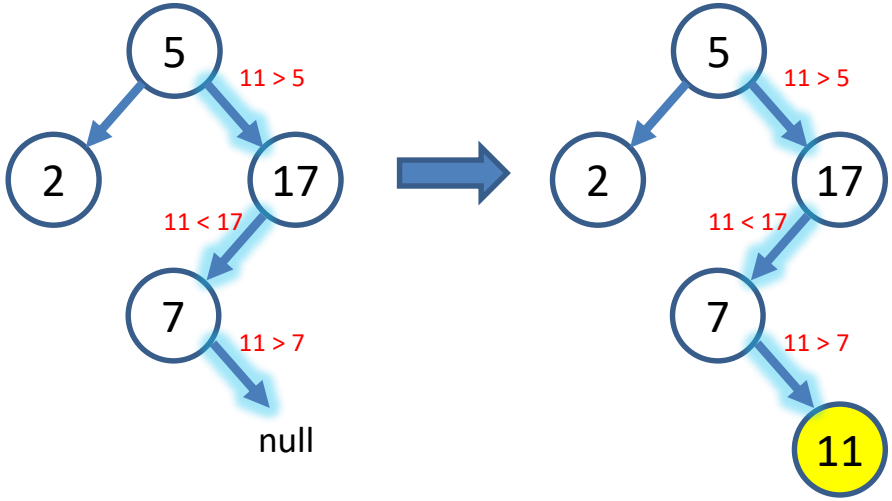




VSTAVLJANJE V BST

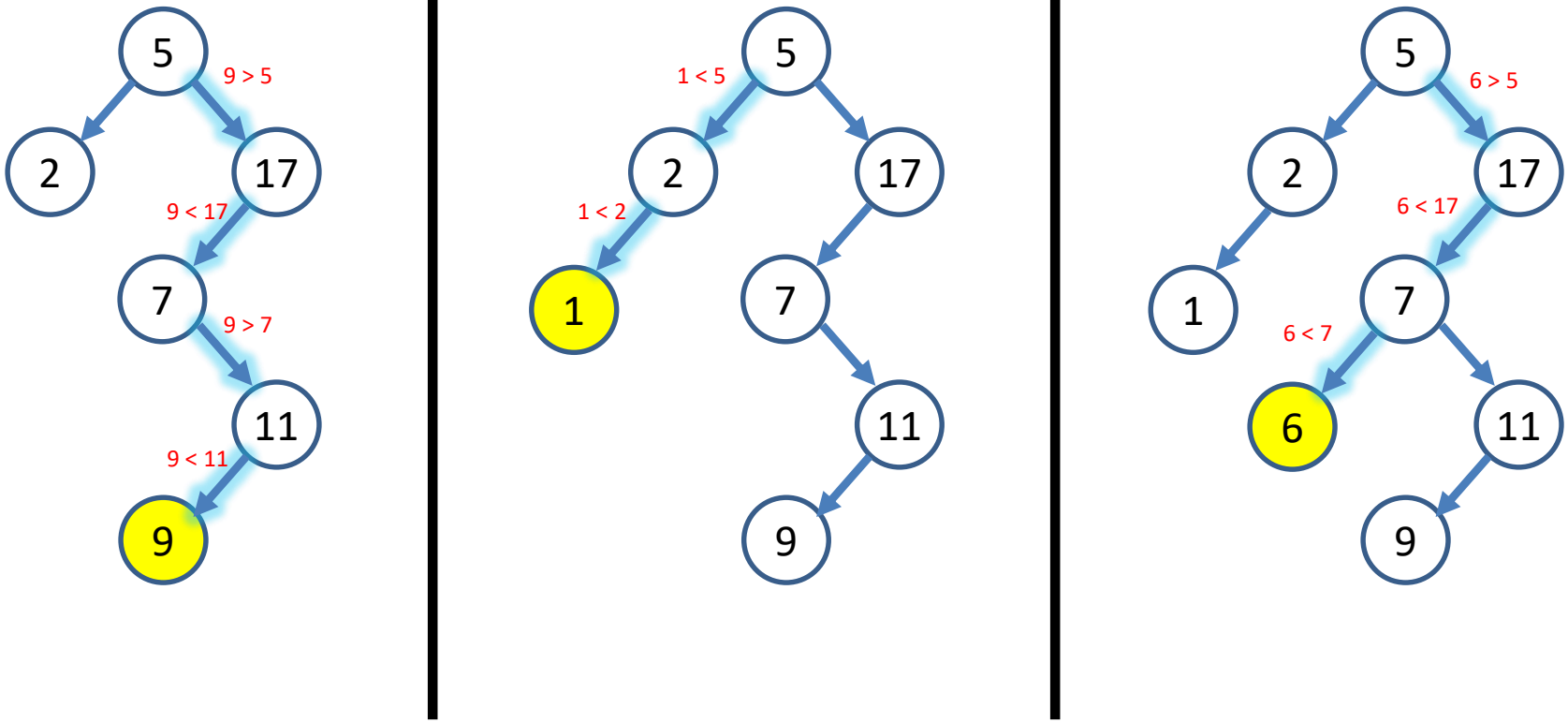
Na začetku je binarno iskalno drevo prazno. Nariši drevo po končanem vstavljanju elementov: 5, 17, 7, 2, 11, 9, 1, 6.





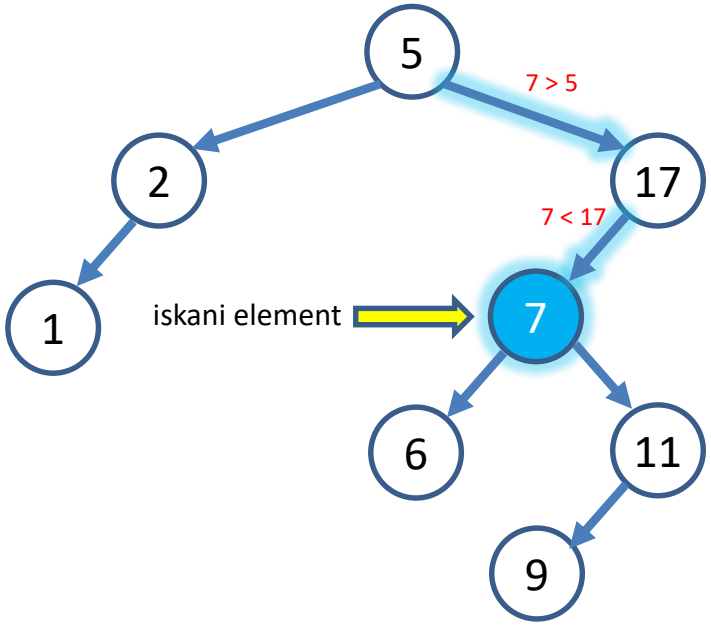
VSTAVLJANJE V BST

Na začetku je binarno iskalno drevo prazno. Nariši drevo po končanem vstavljanju elementov: 5, 17, 7, 2, 11, 9, 1, 6.

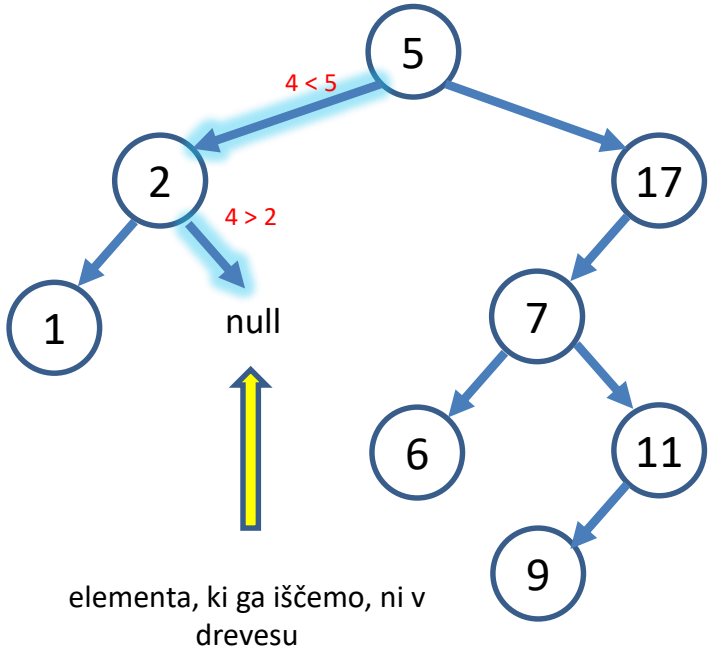


ISKANJE ELEMENTOV V BST

Poišči element 7.

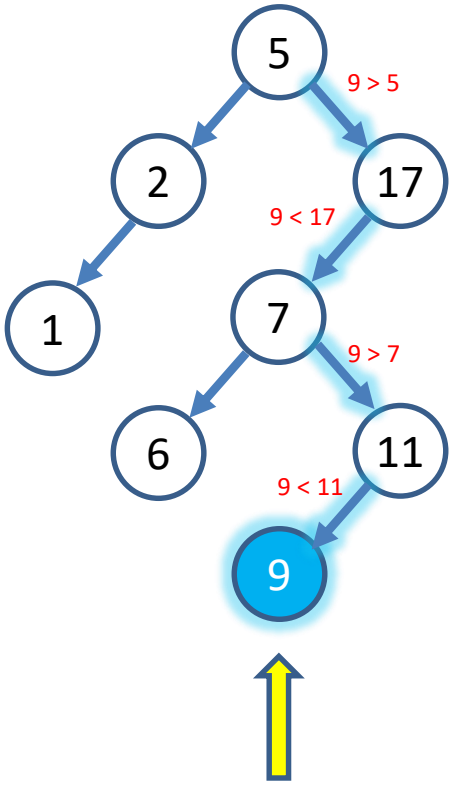


Poišči element 4.

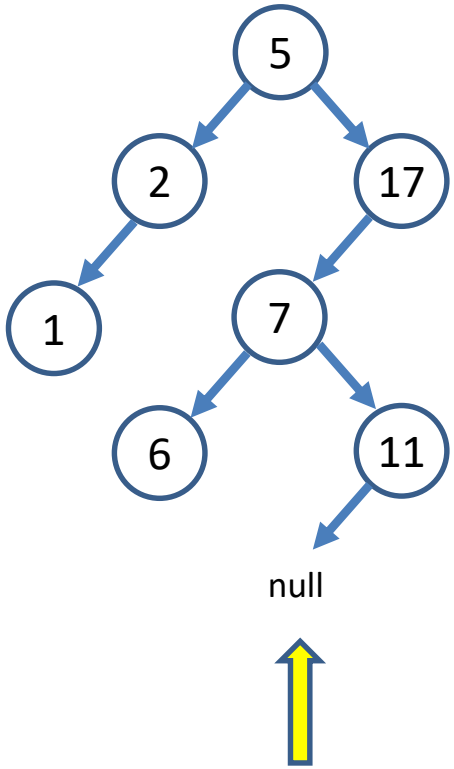


BRISANJE ELEMENTOV IZ BST

Iz drevesa briši elemente: 9, 17,5.



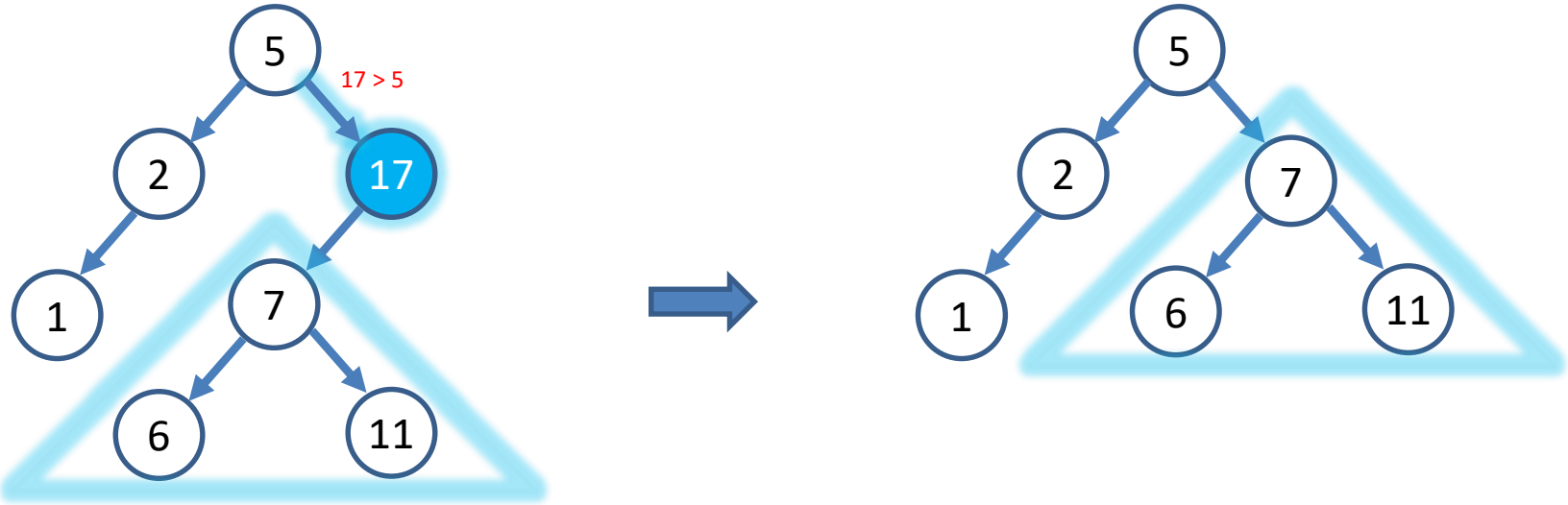
najprej poiščemo element, ki ga želimo brisati



če brišemo element v listu drevesa, ga preprosto odstranimo

BRISANJE ELEMENTOV IZ BST

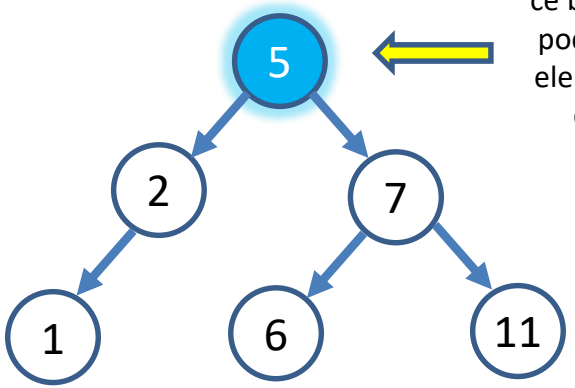
Iz drevesa briši elemente: 9, 17,5.



če brišemo element v notranjem vozlišču, ki ima samo eno poddrevo,
ga nadomestimo s tem poddrevesom

BRISANJE ELEMENTOV IZ BST

Iz drevesa briši elemente: 9, 17,5.

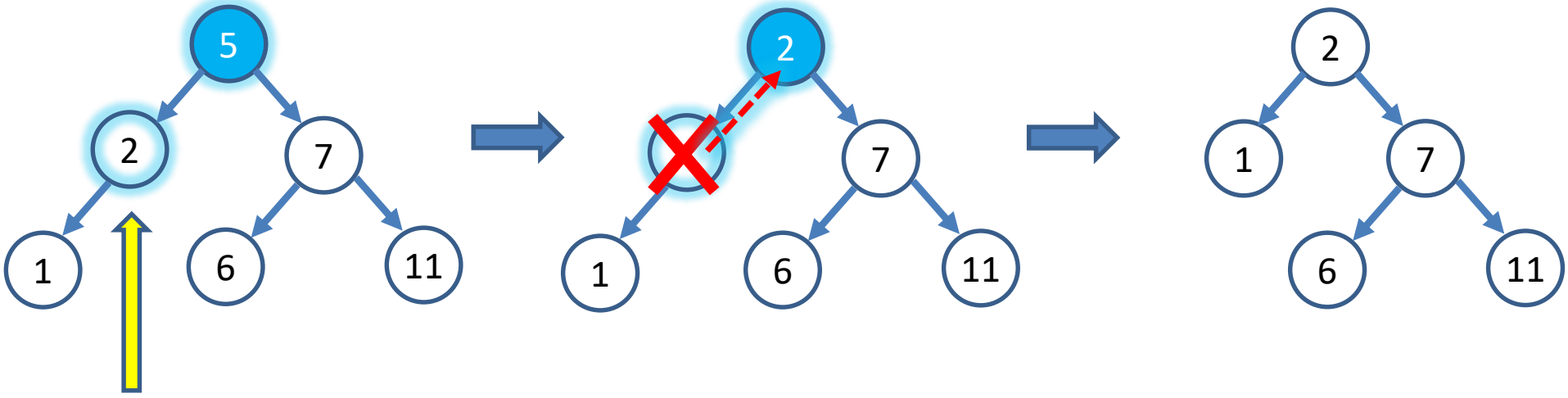


če brišemo element v notranjem vozlišču, ki ima obe poddrevesi, ga nadomestimo bodisi z **maksimalnim** elementom **levega** poddrevesa bodisi z **minimalnim** elementom **desnega** poddrevesa, nato pa tisti element odstranimo iz drevesa

BRISANJE ELEMENTOV IZ BST

Iz drevesa briši elemente: 9, 17,5.

Prvi način brisanja elementa 5:

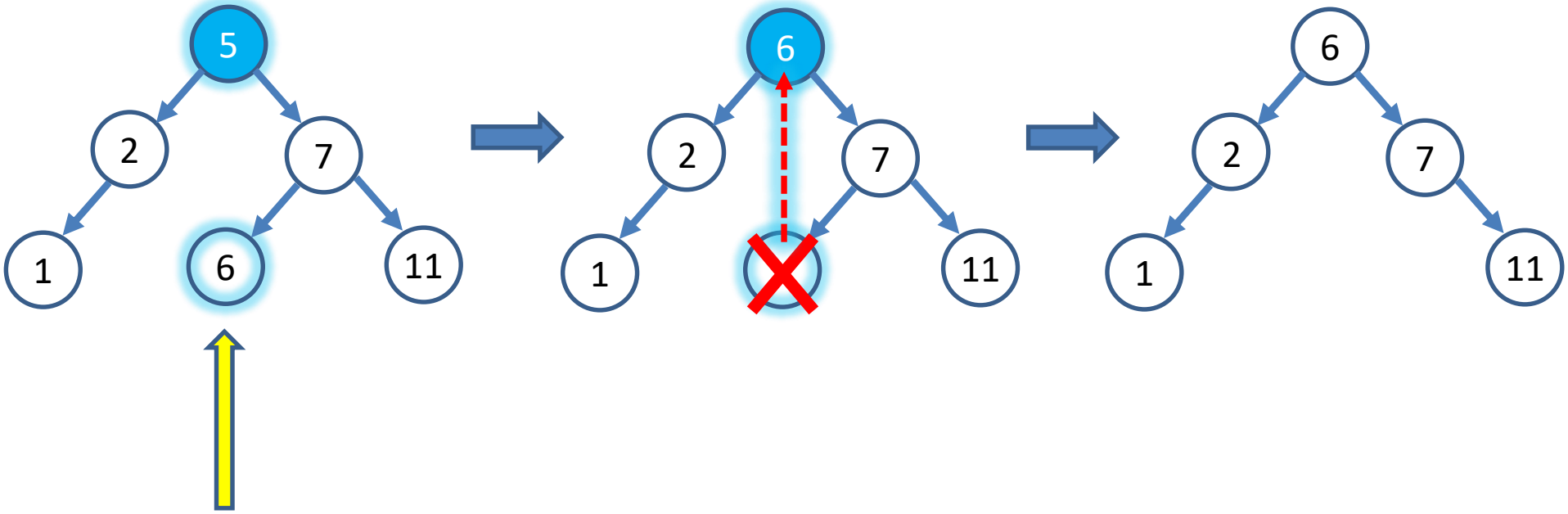


maksimalni element
levega poddrevesa

BRISANJE ELEMENTOV IZ BST

Iz drevesa briši elemente: 9, 17,5.

Drugi način brisanja elementa 5:



NALOGE

Implementirajte naslednje metode v razredu BSTree:

- `void prune()` – poreže liste drevesa
- `int height()` – vrne višino drevesa
- `boolean isBalanced()` – preveri, ali je drevo (delno) uravnovešeno
- `int numOfElements()` – vrne število elementov v drevesu
- `void insertIter(Comparable k)` – iterativno vstavi element v list drevesa
- `boolean iterMember(Comparable k)` – iterativno preveri, ali se podani element nahaja v drevesu
- `void descending()` - izpiše element drevesa v padajočem vrstnem redu
- `BSTNode predecessor(Comparable k)` – vrne kazalec na element drevesa s prvo manjšo vrednostjo od podanega elementa
- `BSTNode successor(Comparable k)` – vrne kazalec na element drevesa s prvo večjo vrednostjo od podanega elementa

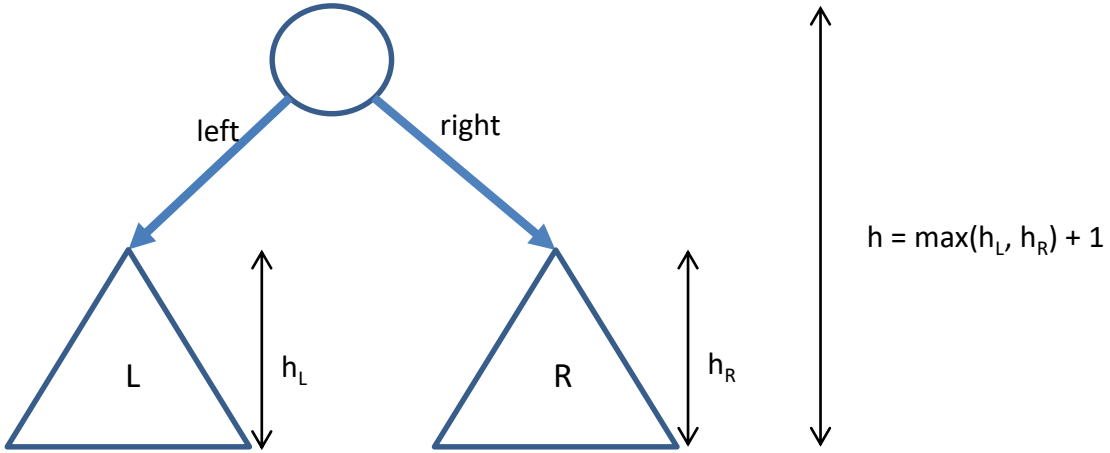
Višina drevesa je dolžina najdaljše poti od korena do lista drevesa.

Binarno iskalno drevo je (delno) **uravnovešeno**, če za vsako vozlišče velja, da se višini obeh poddreves razlikujeta največ za 1.

BINARNO ISKALNO DREVO

`int height()` – vrne višino drevesa

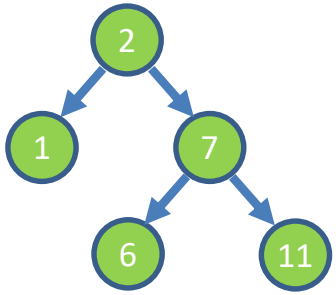
- Višina drevesa je dolžina najdaljše poti od korena do lista drevesa.
- Višina praznega drevesa je 0.

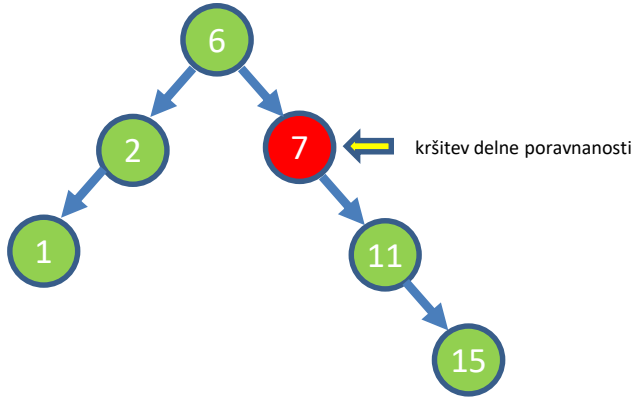


BINARNO ISKALNO DREVO

`boolean isBalanced()` – preveri, ali je drevo (delno) uravnovešeno

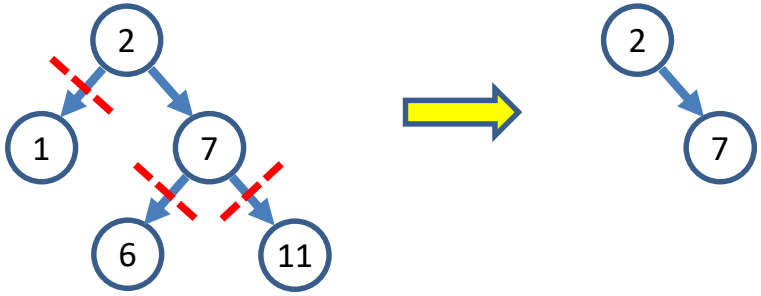
- Binarno iskalno drevo je (delno) **uravnovešeno**, če za vsako vozlišče velja, da se višini obeh poddreves razlikujeta največ za 1.





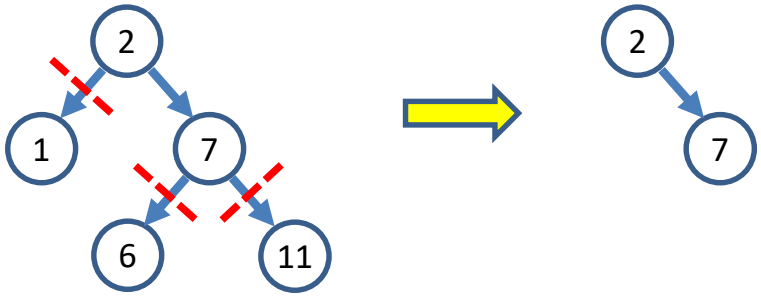
BINARNO ISKALNO DREVO

`void prune()` – poreže liste drevesa



BINARNO ISKALNO DREVO

`void prune()` – poreže liste drevesa




```
public void prune()
{
    root = prune(root);
}


protected BSTNode prune(BSTNode node)
{
    if (node == null)
        return null;

    if (node.left == null && node.right == null)
        return null;

    node.left = prune(node.left);
    node.right = prune(node.right);

    return node;
}
```