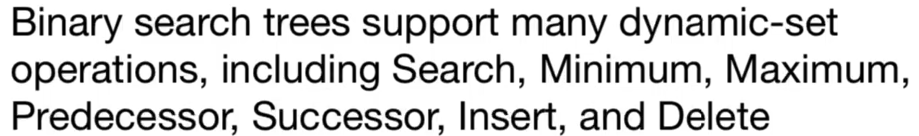
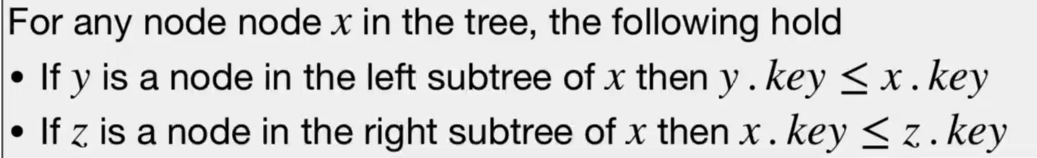
**Binary Search Trees**



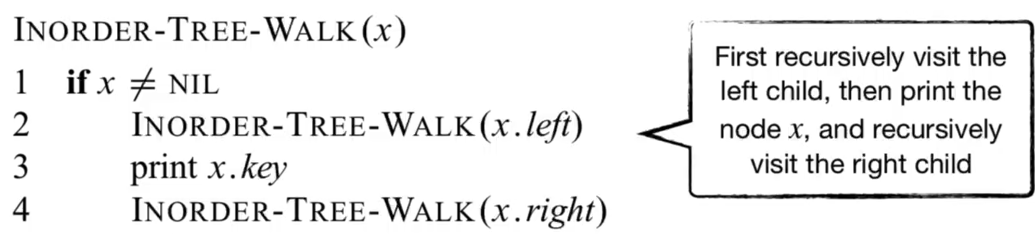
Binary search tree er opbygget som en hvert andet træ. Dog skal den opretholde Binary-search-tree-property:



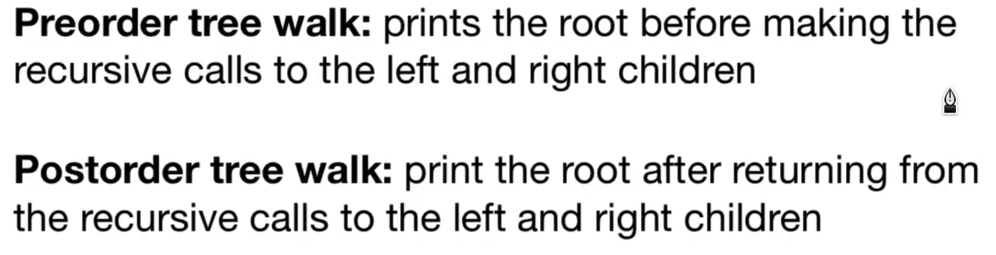
* Altså en node til venstre er altid størrer end dens højre modpart.

**Operationer på BST’s**

**Inorder** tree walk bliver brugt til at printe alle keys i træet i sorteret rækkefølge.

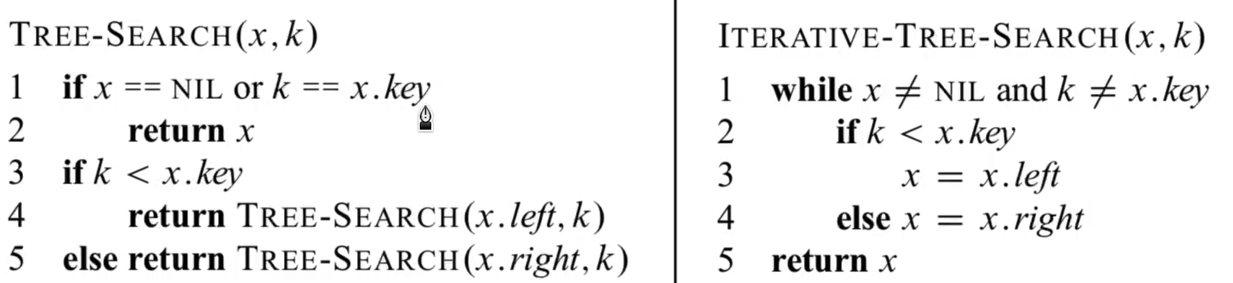


Der findes også andre variationer af algoritmen:



Alle tree-walks tager theta(n) tid.

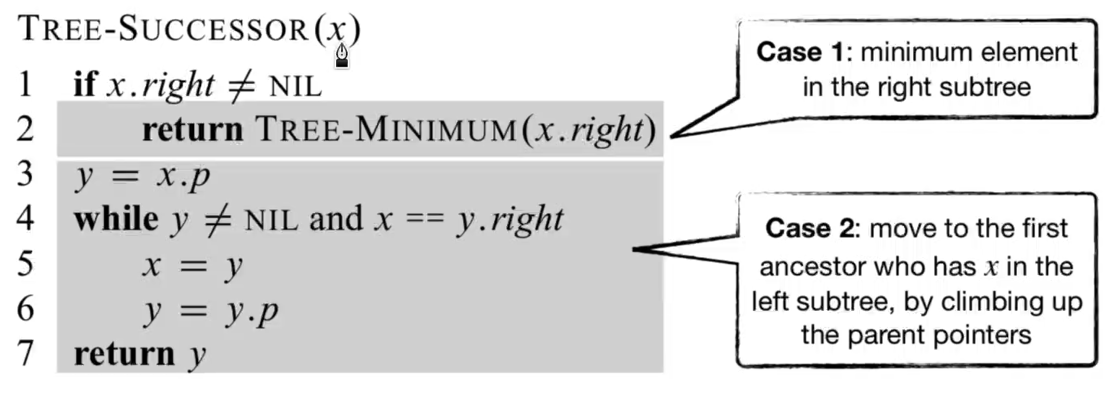
**Tree-search** findes i 2 varianter: rekursiv og iterativ. Worstcase runtime for begge er O(h) hvor h er højden af træet.



**Tree minimum og tree-maximum** returnere det største/mindste element i træet. Kørertiden er O(h). Her bruges Binary-search-tree-property, ligesom de andre algoritmer.



**Successor og predecessor**

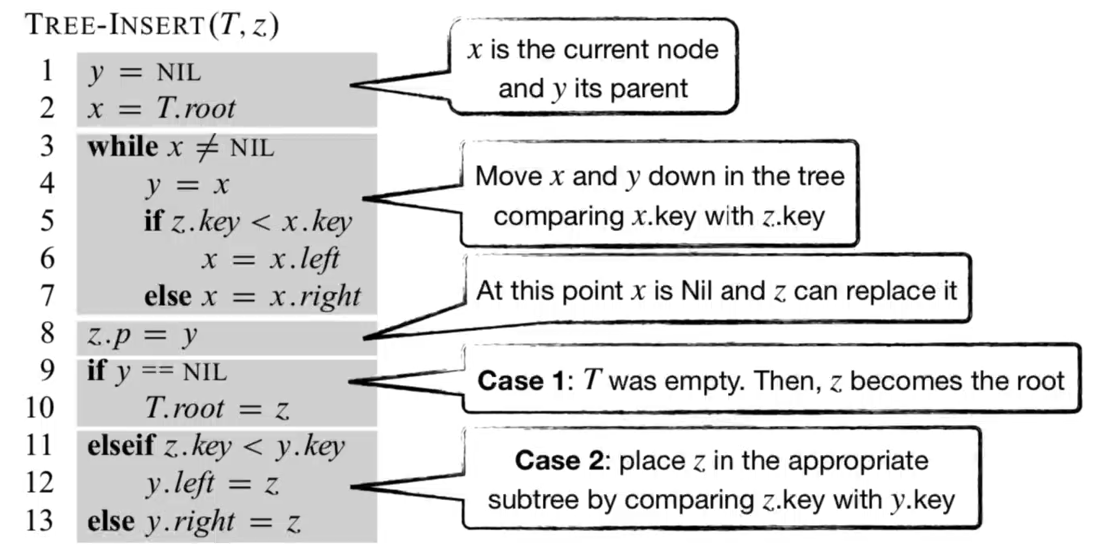


Predecessor er modsat.

Kørertiden: theta(h).

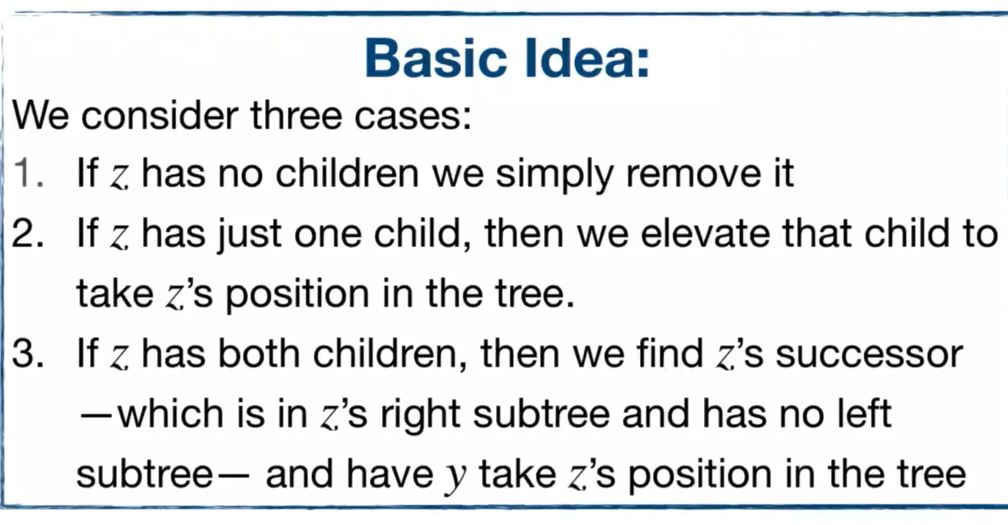
**Tree-insert**

Indsættelsen af et element i træet, hvor binary-tree-property skal overholdes.

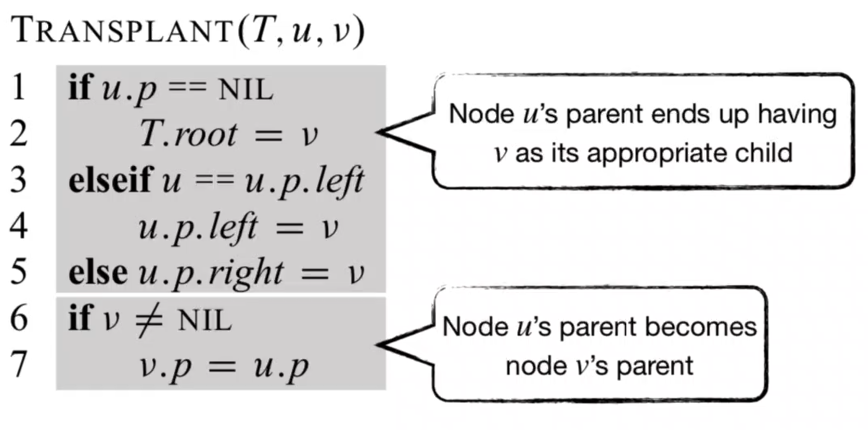


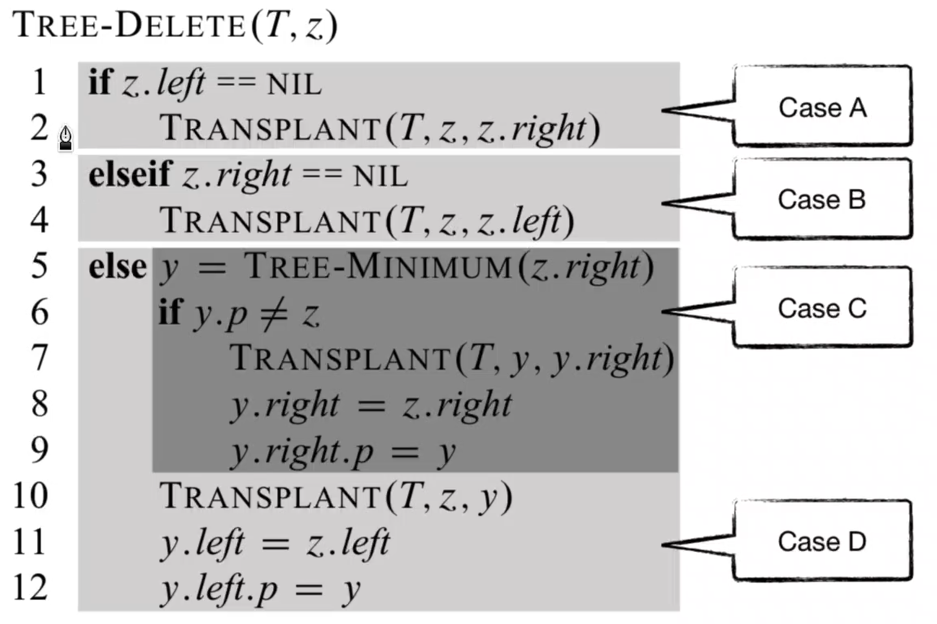
Kørertiden er theta(h).

**Tree-delete**



Tree-delete bruger en hjælpefunktion: Transpant til at erstatte sub-træer:



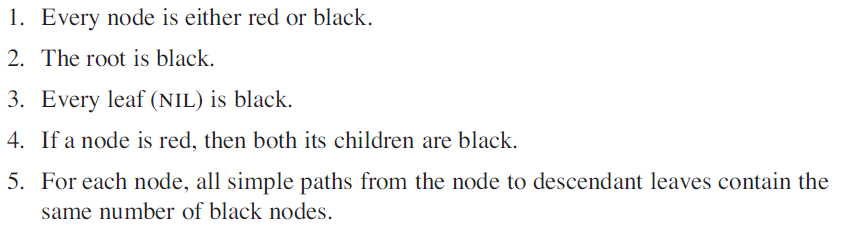


De 4 cases er beskrevet i god detalje i bogen ved BST kapitlet.

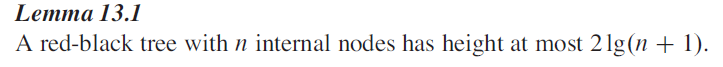
**Red-Black trees**

Et red-black tree er et BST med 1 extra bit til farve. Her kan hver node være enten RED eller BLACK.

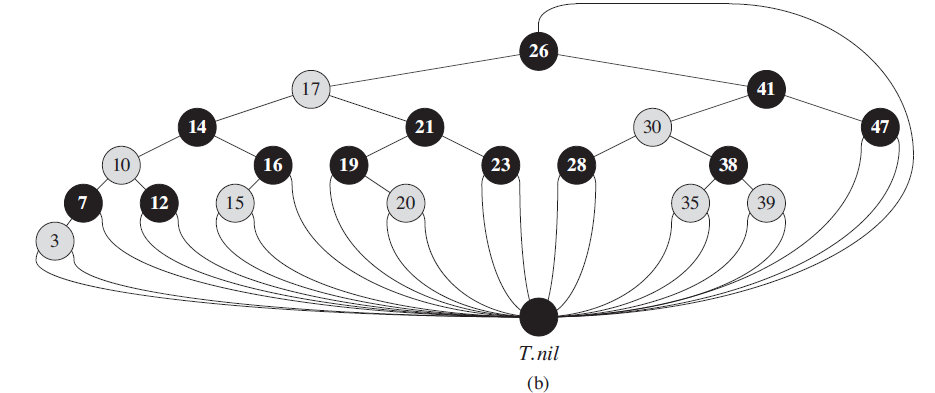
Et Red-black træ skal overholde følgende egenskaber:



Alle pointers til NIL, pointer til T.NIL for boundry conditions.



Eksempel på et red-black tree:



Intro video: <https://www.youtube.com/watch?v=qvZGUFHWChY>

**Rotations**

Rotationer sker når træet bliver ændret og red-black properties skal opretholdes.

<https://www.youtube.com/watch?v=95s3ndZRGbk>

**Insertion**

<https://www.youtube.com/watch?v=5IBxA-bZZH8&t=5s>

<https://www.youtube.com/watch?v=A3JZinzkMpk>

**Deletion**

<https://www.youtube.com/watch?v=BIflee1rLDY&t=12s>