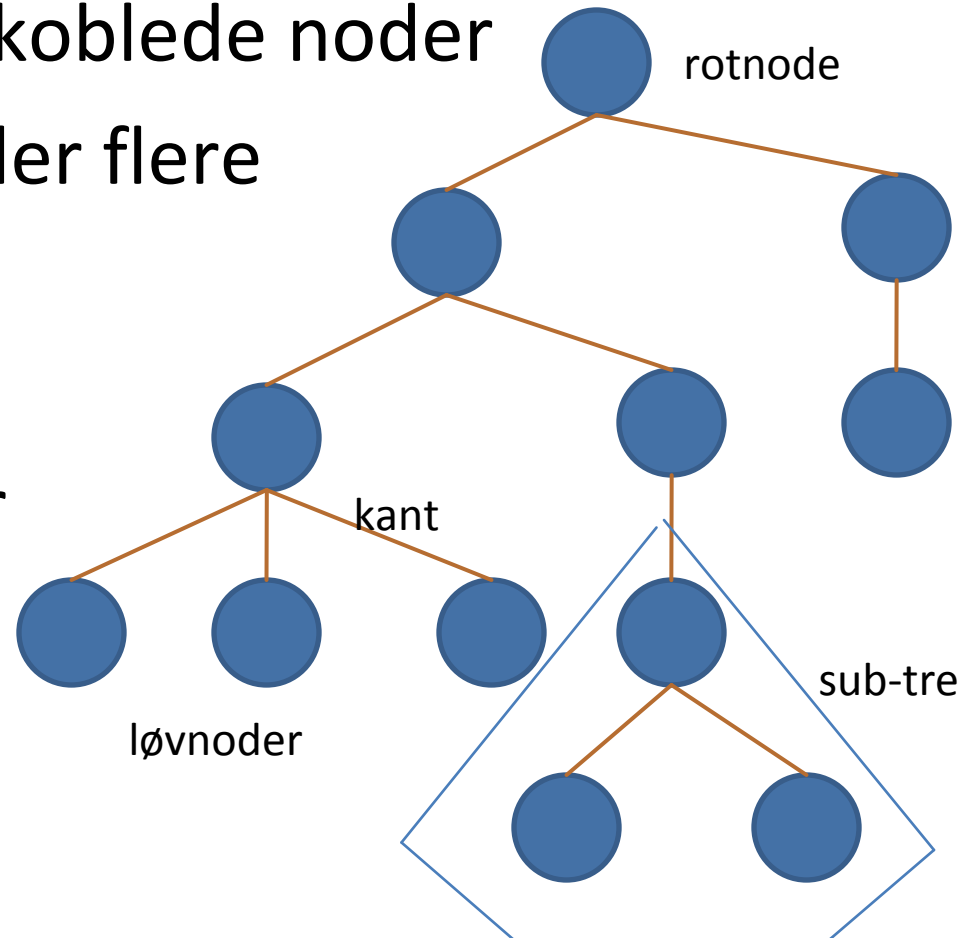


Trær

- Består av sammenkoblede noder
- Hver node har 0 eller flere barne-noder.
- Må være asyklisk.
- Et tre med n noder har $n-1$ kanter.

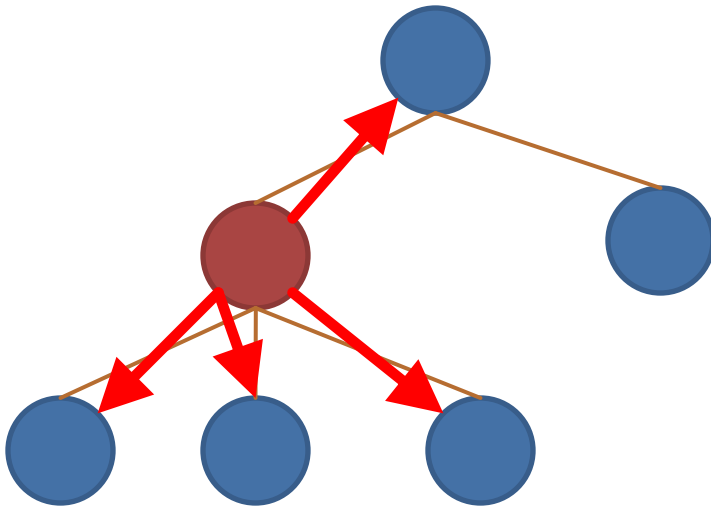


Oppbygging

Variant 1:

Noden har kjenskap til:

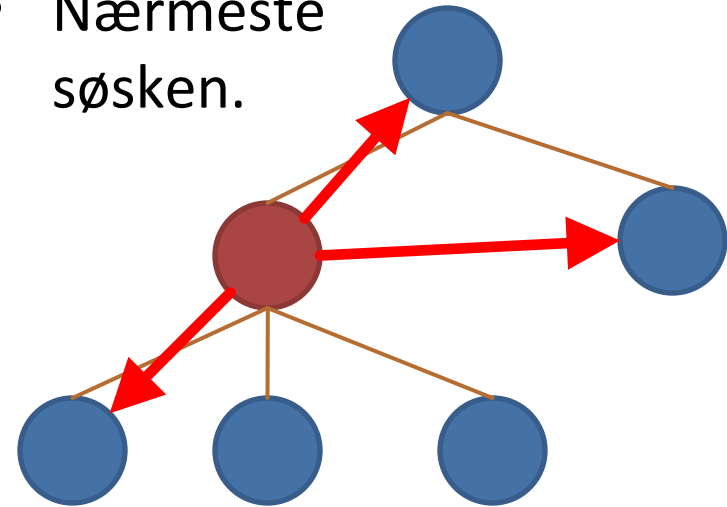
- Foreldrenoden
- Alle barnenodene.



Variant 2:

Noden har kjenskap til:

- Foreldrenoden
- Venstre barnenode.
- Nærmeste søsken.

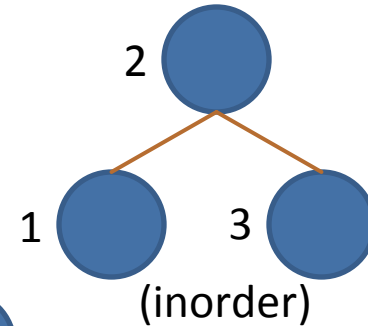


(kjenskap er markert med rødt)

Traversering

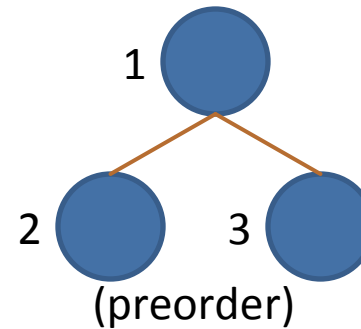
Inorder:

- Venstre subtre behandles før rotnoden.
Deretter høyre subtre.



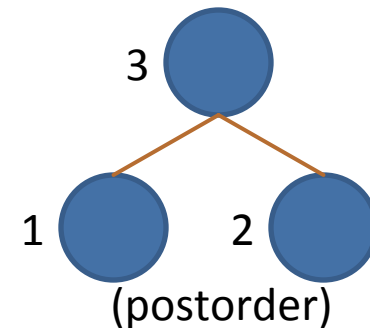
Preorder:

- Rotnoden behandles først.
Deretter de to subtrærne.



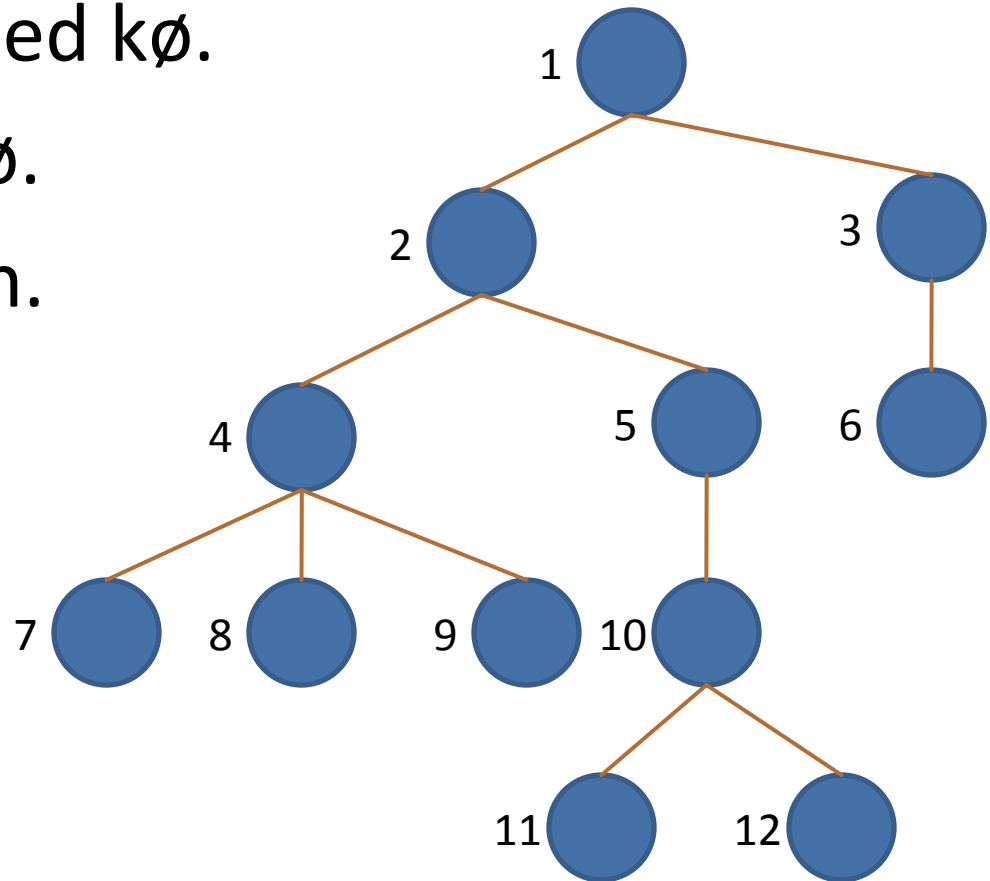
Postorder:

- De to subtrærne behandles først.
Deretter behandles rotnoden.



Bredde først søk

- Implementeres med kø.
- Putt rotnoden i kø.
- Pop node fra køen.
Putt alle barna på køen. Gjenta til køen er tom.

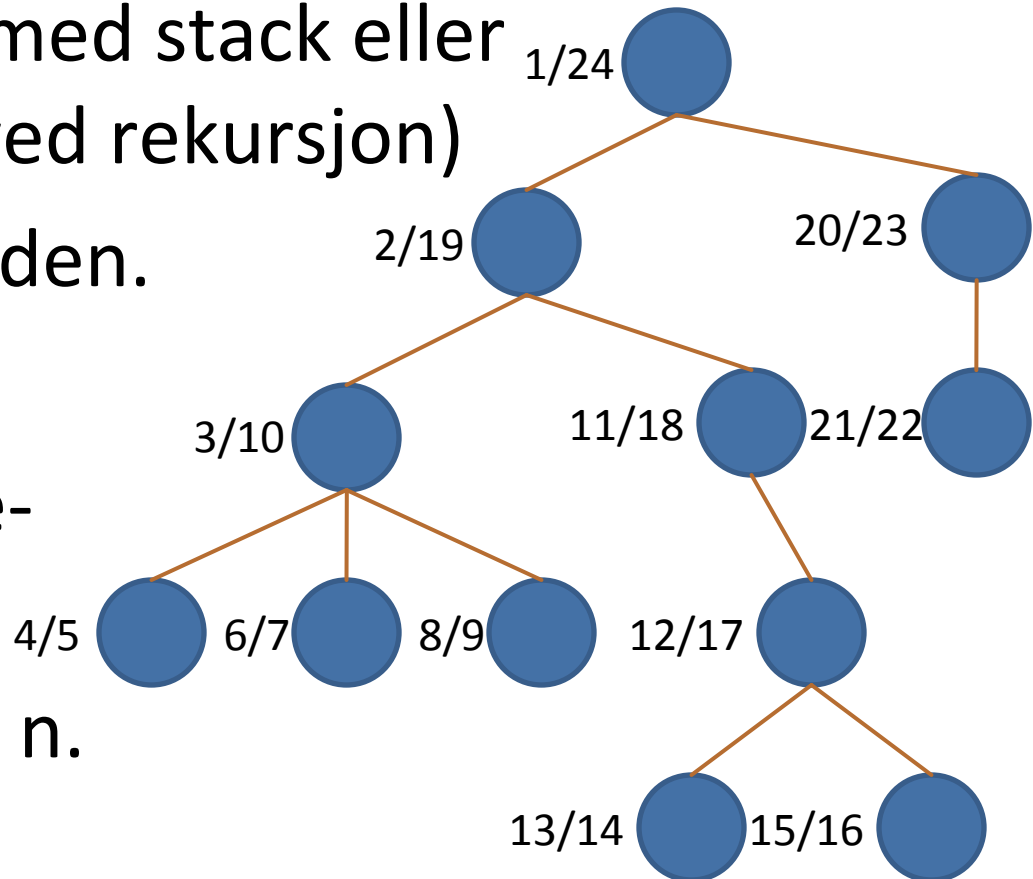


Dybde først søk

- Implementeres med stack eller rekursjon. (her ved rekursjon)
- Kall dfs på rotnoden.

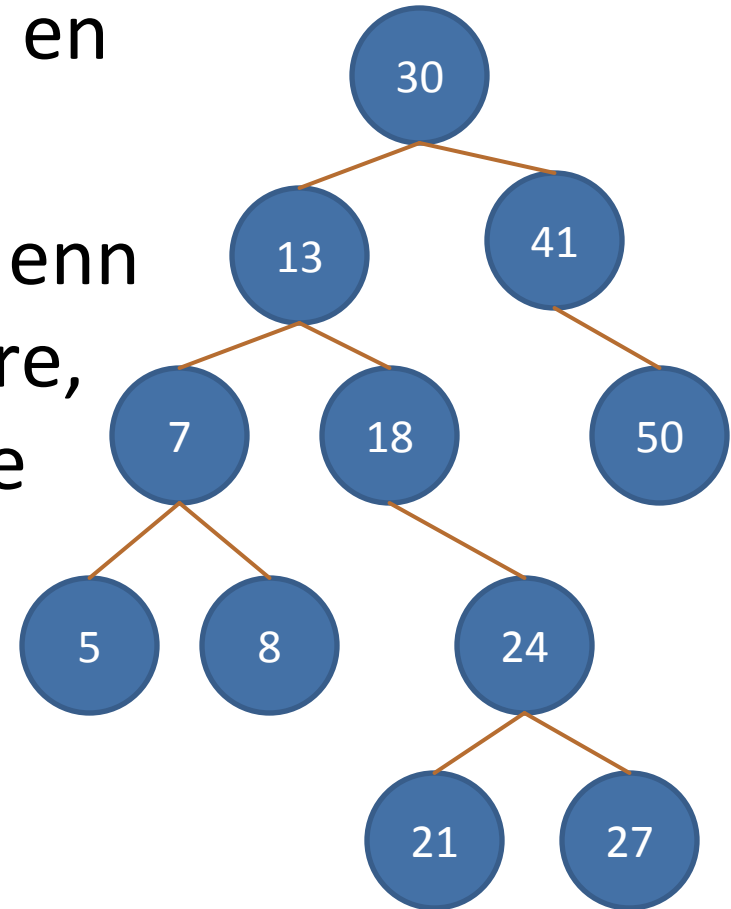
dfs(node n):

- Kall dfs på barne-
nodene
- Behandle noden n.



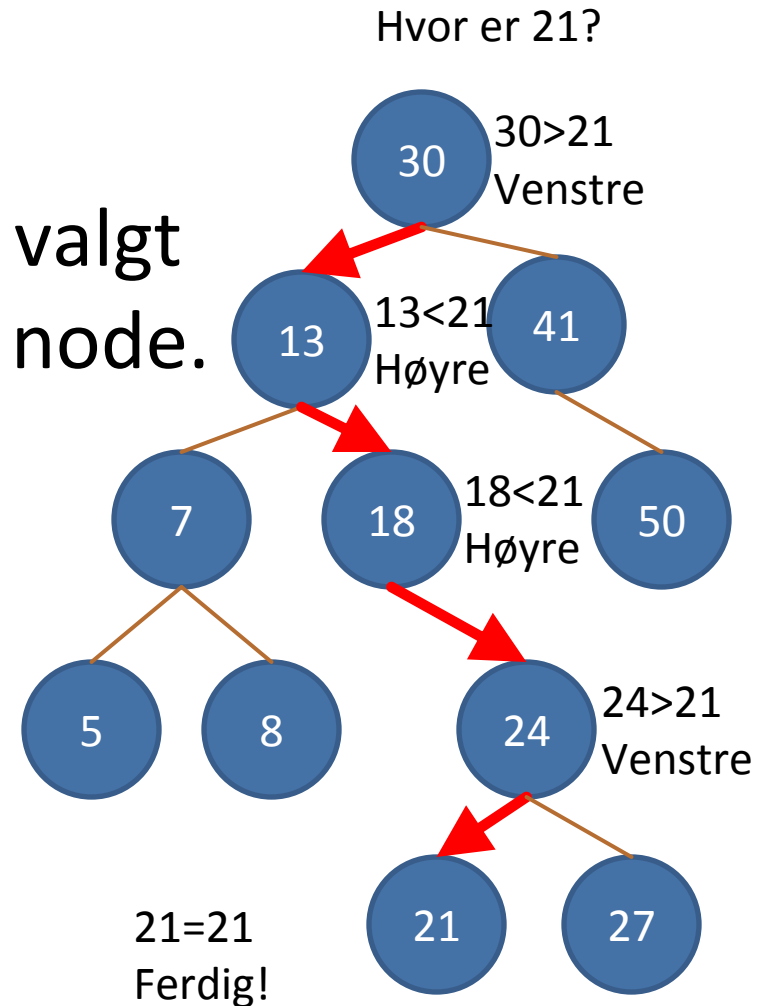
Binære søketrær

- Hver node kan ikke ha flere en 2 barn.
- Verdien i en node er større enn alle verdiene i venstre subtre, og mindre enn alle verdiene i høyre subtre



Søk

- Start i rotnoden
- Gå til venstre hvis verdien i valgt node er høyere enn ønsket node. Høyre hvis motsatt.
- Gjenta til ønsket node er funnet, eller det ikke er mulig å gå lenger.



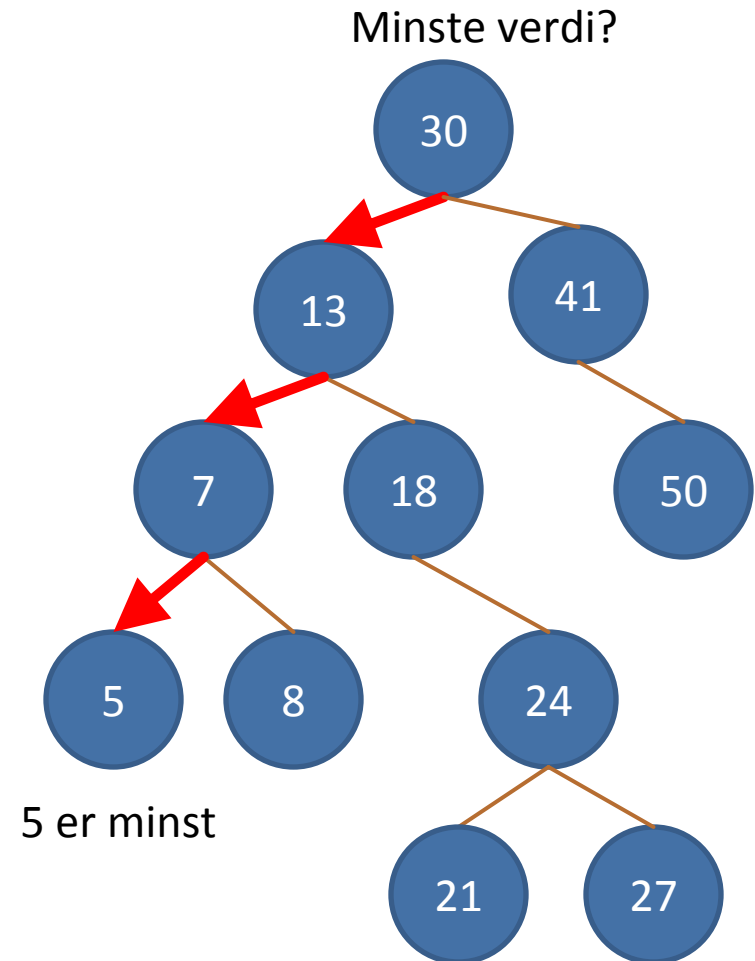
Minimum/Maksimum

Minimum:

- Start i rotnoden.
- Velg venstre barn.
- Gjenta til du kommer til en løvnode. Den har minste verdi.

Maksimum:

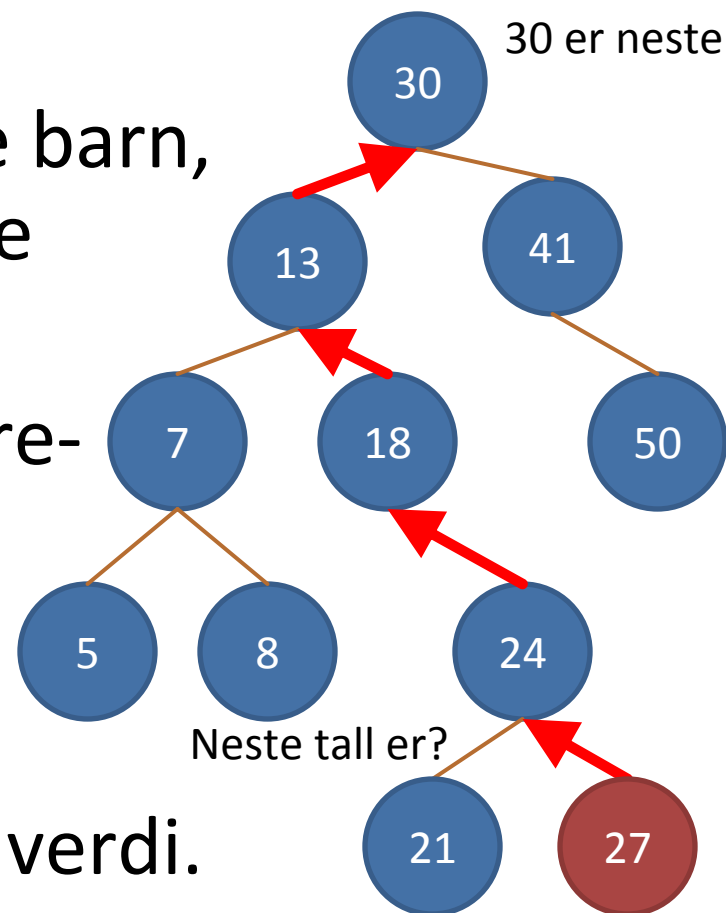
- Symetrisk av minimum.



Foregående/etterfølgende verdi

Etterfølgende verdi:

- Dersom noden har et høyre barn, finn minste verdi i det høyre sub-treet.
- Hvis ikke, finn første foreldre-noden som ikke har forrige node som høyre barn.

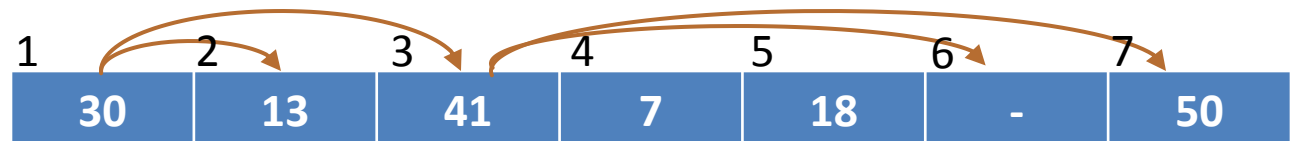
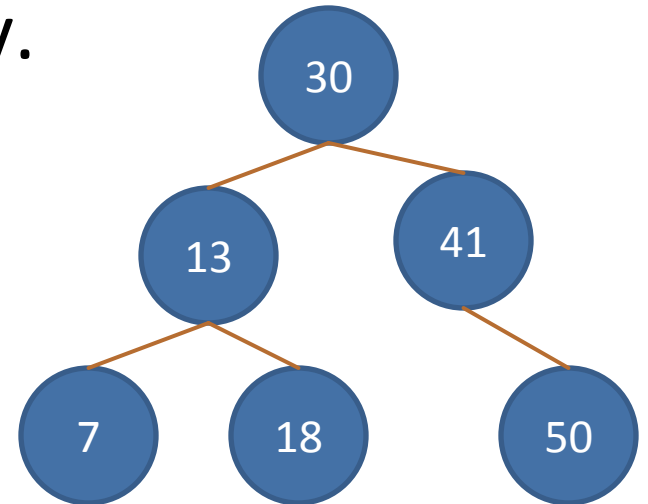


Foregående verdi:

- Symetrisk av etterfølgende verdi.

Implementasjon

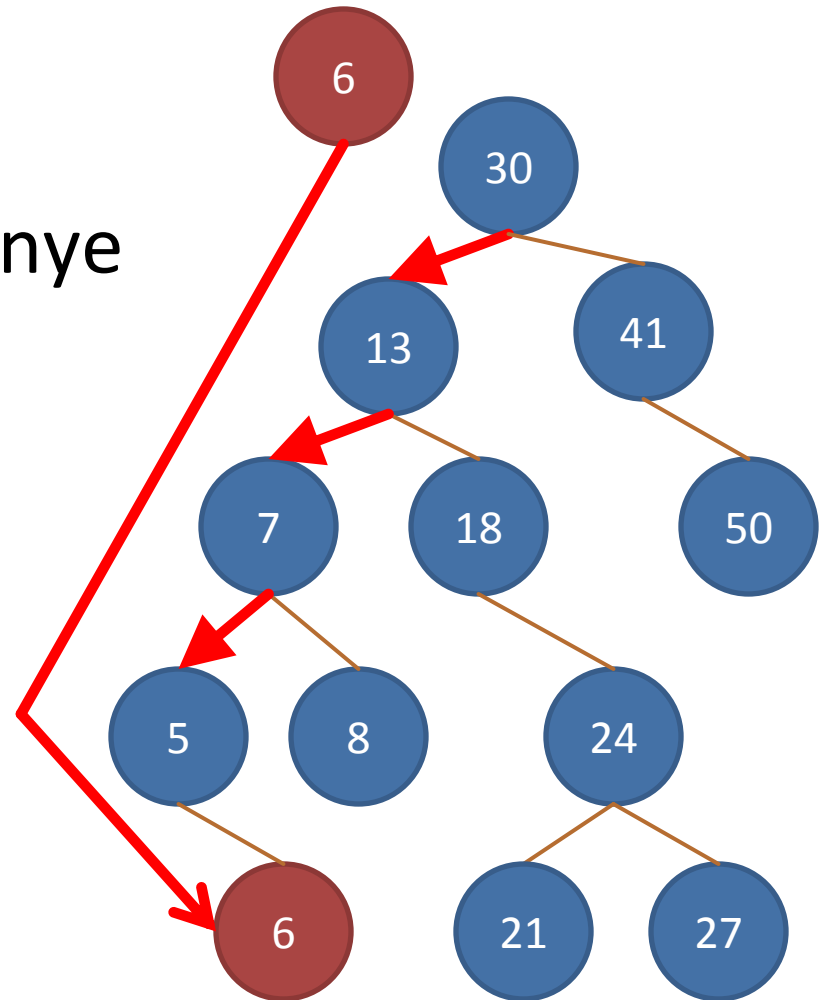
- Kan implementeres med array.
- Slektskap til node i :
Foreldre: $\lfloor i / 2 \rfloor$
Venstre barn: $2i$
Høyre barn: $2i + 1$
- Se figur 6.1 (s 128)



VIKTIG: Tabellen starter på 1.

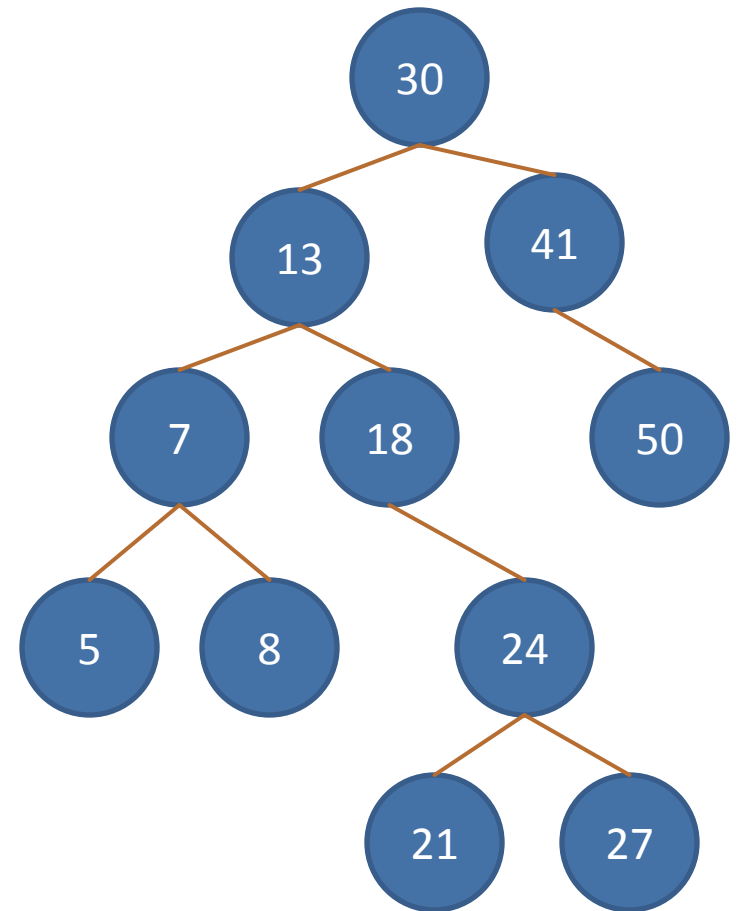
Innsetting

- Fungerer veldig likt søk.
- Søk etter verdien til den nye noden.
- Plasser noden som venstre/høyre barn avhengig av verdien.



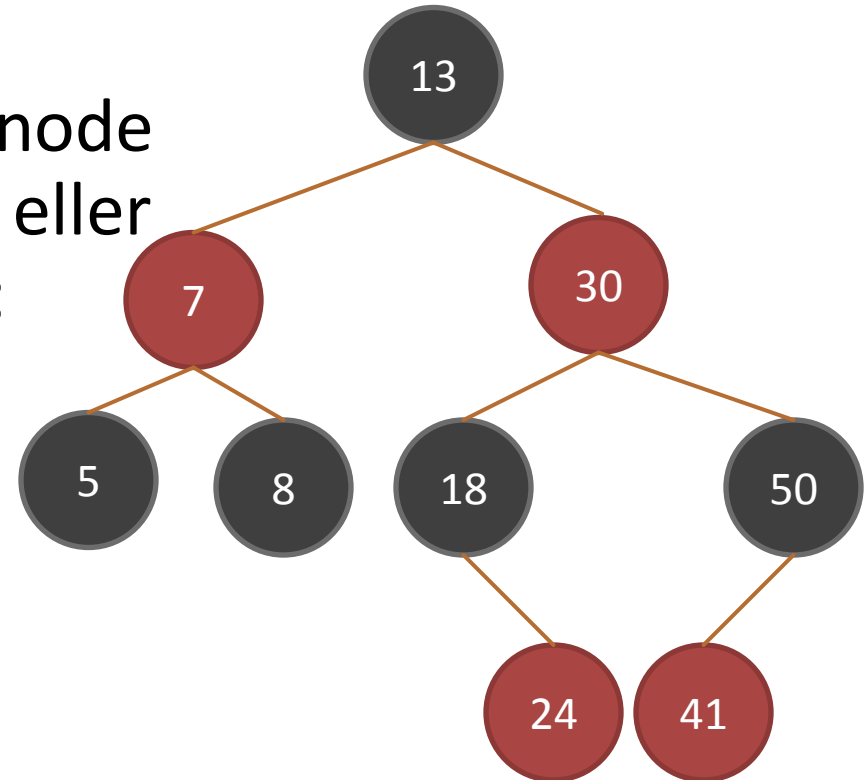
Sletting

- Se side 262.



Rød-svarte trær

- Variant av binært tre.
- I denne tretypen får hver node tildelt en farge, enten rød eller svart, basert på tre regler:
 - Roten i treet er svart
 - Om en node er rød, må barna være svarte
 - Alle stier fra en node til en løvnode må inneholde det samme antallet svarte noder.
- Garanterer en makshøyde på $2\log(N+1)$



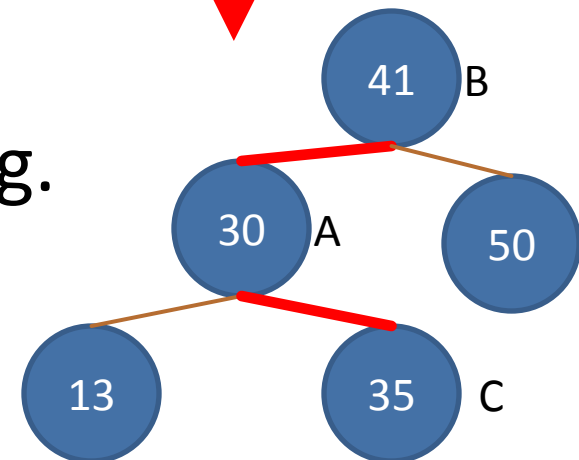
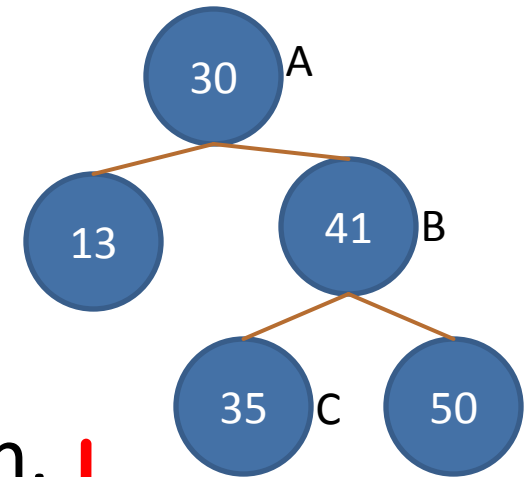
Rotasjon

Venstre-rotering:

- Sett node A som B sitt venstre barn.
- Sett node C som A sitt høyre barn.

Høyre-rotering:

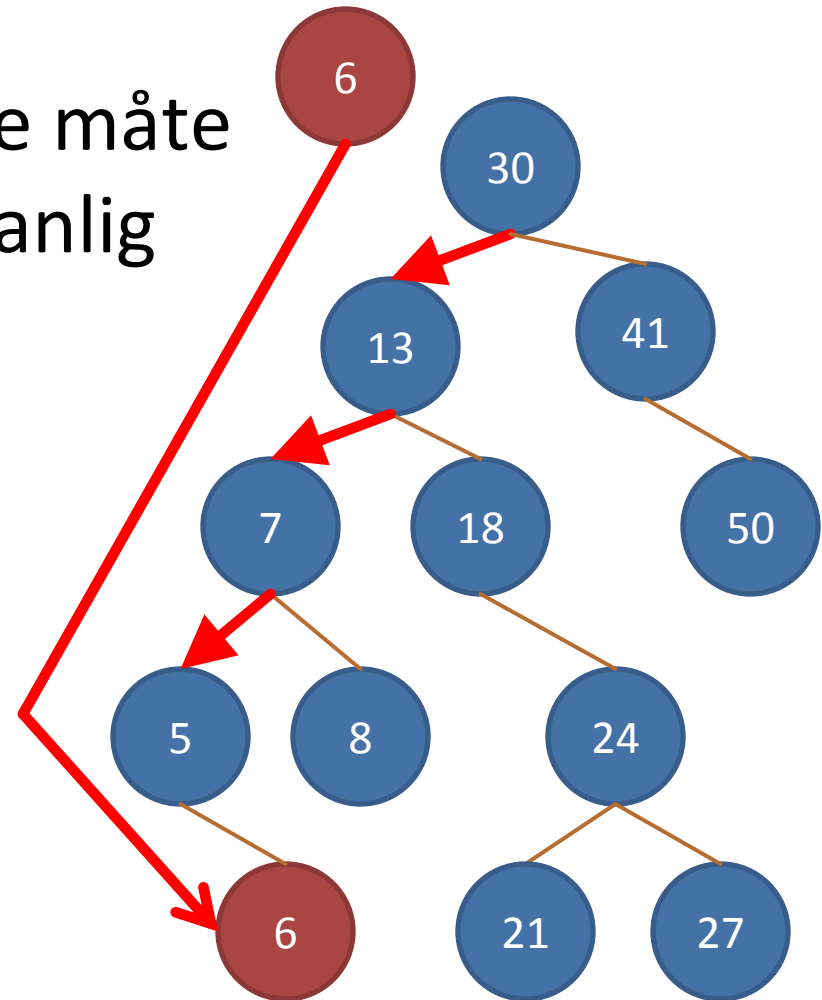
- Symetrisk av venstre-rotering.



(røde kanter er endrede kanter)

Innsetting

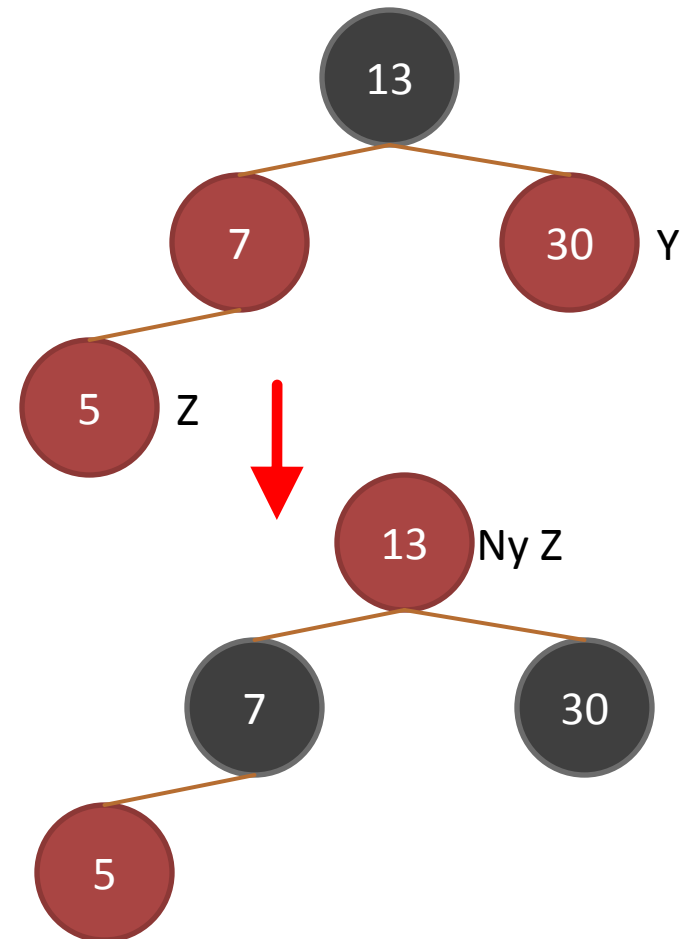
- Sett inn noden på samme måte som ved innsetting i et vanlig binær-tre.
- Farg noden rød.
- Rydd opp!



Opprydding (1)

Case 1: Z's onkel Y er rød.

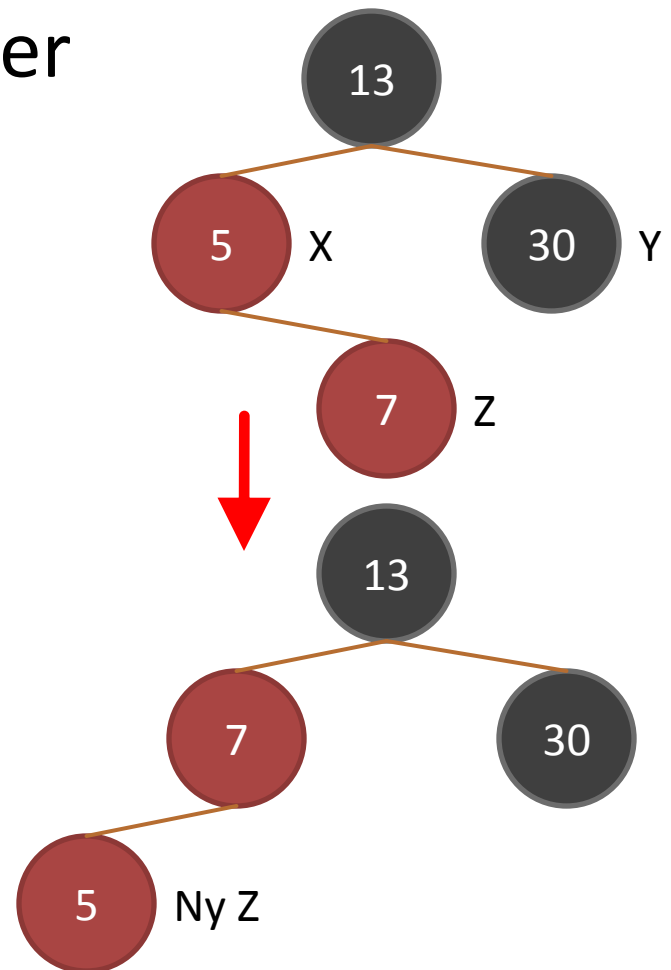
- Farg Z's far og onkel svarte.
- Farg Z's bestefar rød.
- Fortsett oppryddning fra Z's bestefar.



Opprydding (2)

Case 2: Z's onkel Y er svart, og Z er et høyre barn.

- Venstreroter Z og X for å få endret til case 3.
- Fortsett med "case 3"-opprydding.



Opprydding (3)

Case 3: Z's onkel Y er svart, og Z er et venstre barn.

- Høyre-roter X og W
- Farg X svart
- Farg W rød.

