# *Uke 4,*
# *Forelesning 2*







- Essensen i faget: Effektive algoritmer
- Matematiske forutsetninger
- Introduksjon til metodekall og rekursjon
- Introduksjon til kombinatoriske søk, kombinasjoner og permutasjoner
- Introduksjon til analyse av algoritmer

Uke 1 og Uke 2

- Abstrakte datatyper (ADT'er)
- Lister, stabler og køer

Uke 3 og Uke 4.1



## OVERSIKT – *Uke 4, Forelesning 2 (W4.L2)*

- Vi begynner nytt tema: TRÆR
  - DENNE GANGEN:
    - TEMA #1: Generelle trær (MAW kapittel 4.1)
    - TEMA #2: Binære trær (MAW kapittel 4.2)
    - TEMA #3: Binære søketrær, introduksjon (MAW kap. 4.3)
  - NESTE UKE:
    - TEMA #3: Binære søketrær, fortsetter (MAW kapittel 4.3)
  - OM CA. 2 UKER (uke 6 forelesning 2 eller W6.L2):
    - TEMA #4: B-trær (MAW kapittel 4.7)

- Aktuelle temaer er som for andre ADT'er:
  - Datastrukturen, implementasjonen(e),
  - innsetting/fjerning av elementer, søking etter elementer, traversering,
  - tidsforbruk,
  - bruksområder…

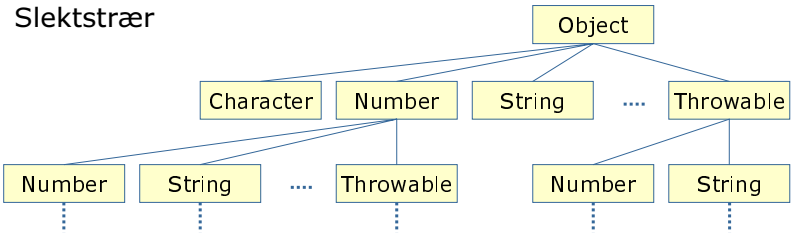


## TEMA #1: TRÆR

*Generelt om generelle trær...*

- Trær brukes gjerne til å representere data eller systemer som er organisert i form av en hierarki.

- Typiske eksempler er:
  - Klassehierarki (i for eksempel Java)
  - Filorganisering
  - Organisasjonskart
  - Slektstrær

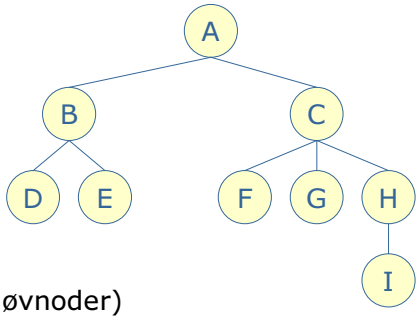







## Definisjon/terminologi

- Et tre er en samling noder. Et ikke-tomt tre består av en spesiell node, kalt roten, og null eller flere ikke-tomme subtrær. Fra roten går det en rettet kant til (roten i) hvert subtre.




## Eksempel/terminologi

- A er roten (rot-noden)
- B er forelder til D og E
- D og E er barna til B
- C er søsken til B
- D, E, F, G og I er bladnoder (løvnoder)
- A, B, C og H er interne noder

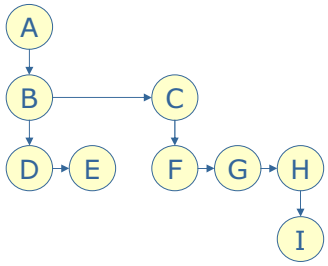
**Forslag 1**

- I tillegg til data kan hver node inneholde en peker til hvert av barna.
- Problemer:
  Antall barn kan variere veldig fra node til node.
  Vet ikke nødvendigvis antall barn på forhånd.

**Forslag 2**

- Hver node inneholder en **liste** med pekere til barna:

```
class TreNode
{ Object element;
  TreNode forsteBarn;
  TreNode nesteSosken;
}
```








- Å traversere et tre vil si å besøke alle (eventuelt bare noen av) nodene i treet, for eksempel fordi vi:
  - Leter etter en bestemt node (element)
  - Vil sette inne en ny node (med et nytt element)
  - Vil fjerne en node/element
  - Vil gjøre en eller annen beregning (eller utskrift) på alle nodene i treet.

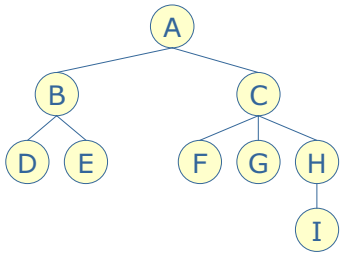
- Rekkefølgen for besøk av noder bestemmes av det problemet vi skal løse.

## To populære traverseringsmåter:

- Prefiks (preorder):
  Gjør noe med en node **før** vi går videre til barna.
- Postfiks (postorder):
  Gjør noe **etter** at vi har besøkt **alle** barna til noden.

## Eksempel

- Prefiks : A B D E C F G H I
- Postfiks : D E B F G I H C A








```
// Datastrukturen
class TreNode
{  Object element;
   TreNode forsteBarn;
   TreNode nesteSosken;
}
```

```
// Prefikstraversering
void prefixTravers()
{  behandleDenneNoden();
   if (forsteBarn != null)
       forsteBarn.prefixTravers();
   if (nesteSosken != null)
       nesteSosken.prefixTravers();
}
```

```
// Postfikstraversering-1
void postfixTravers()
{  if (forsteBarn != null)
       forsteBarn.postfixTravers();
   behandleDenneNoden();
   if (nesteSosken != null)
       nesteSosken.postfixTravers();
}
```

```
// Postfikstraversering-2
void postfixTravers()
{  if (forsteBarn != null)
       forsteBarn.postfixTravers();
   if (nesteSosken != null)
       nesteSosken.postfixTravers();
   behandleDenneNoden();
}
```

- En vei (sti)…
  … fra en node $n_1$ til en node $n_k$
  … er definert som en sekvens av noder $n_1, n_2, \ldots, n_k$
  … slik at $n_i$ er forelder til $n_{i+1}$ for $1 \leq i \leq k$.

- Lengden av denne veien er…
  … antall kanter på veien,
  … det vil si $k$ - 1.

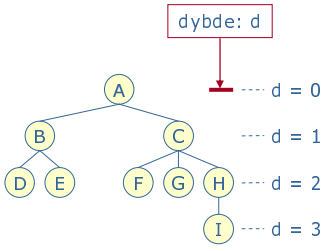




- For enhver node er dybden definert som lengden av (den unike) veien fra roten til noden. Roten har altså dybde 0.

  Rekursiv metode for å beregne dybden til alle nodene i et tre:

```
//   Kall: rot.beregnDybde(0)
int dybde;
void beregnDybde(int d)
{  dybde = d;
   for ( <hvert barn b> )
   {  b.beregnDybde(d + 1);
   }
}
```

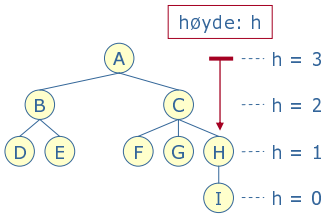



  Dette er et eksempel på **prefiks** traversering

- Høyden til en node er definert som lengden av den lengste veien fra noden til en bladnode. Alle bladnoder har dermed høyde 0. Høyden til et tre er lik høyden til roten.

Rekursiv metode for å finne høyden til alle nodene i et tre:

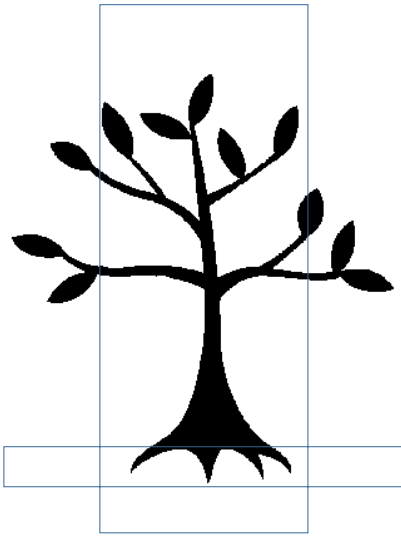
```
// Kall: rot.beregnHoyde()
int hoyde = 0;
int beregnHoyde()
{  int tmp;
   for ( < hvert barn b > )
   {  tmp = b.beregnHoyde() + 1;
      if (tmp > hoyde)
         hoyde = tmp;
   }
   return hoyde;
}
```




Dette er et eksempel på **postfiks** traversering

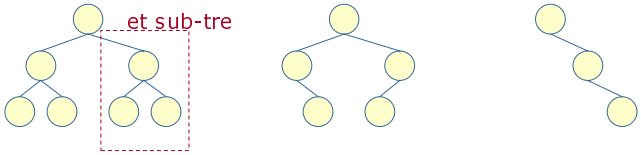






*Om binære trær...*

- Et binært tre er et tre der hver node aldri har mer enn to barn.
- OBS! Det betyr ikke at hver node har eksakt to barn men maksimum to barn, dvs. enten 0 barn, 1 barn eller 2 barn...
- Dersom det bare er ett subtre, må det være angitt om dette er venstre eller høyre subtre.




- I verste fall blir dybden N-1!

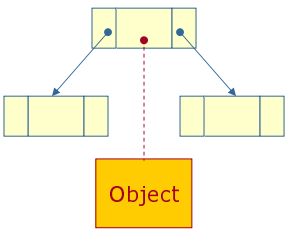




## Implementasjon

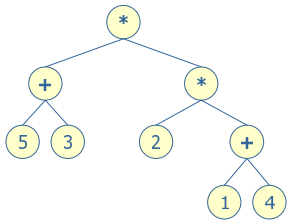
Siden hver node nå har maks to barn, kan vi ha pekere direkte til dem:

```
// Datastruktur for en binærtre-node
class BinNode
{ Object element;
  BinNode venstre;
  BinNode hoyre;
}
```



## Eksempel: Uttrykkstrær

- I uttrykkstrær inneholder bladnodene operander (konstanter, variable, . . . ), mens de interne nodene inneholder operatorer.



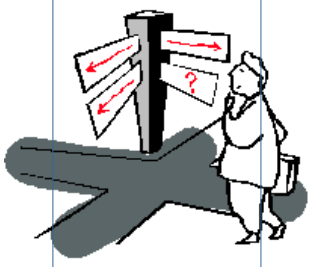

- Mulige skrivemåter for dette uttrykket:
  - Prefiks: * + 5 3 * 2 + 1 4
  - Infiks: (5 + 3) * (2 * (1 + 4))
  - Postfiks: 5 3 + 2 1 4 + * *





## Traversering av binærtrær: oppsummering

```
void traverser( BinNode n )
{ if ( n != null )
  { < Gjør PREFIKS-operasjonene >
    traverser( n.venstre );
    < Gjør INFIKS-operasjonene >
    traverser( n.hoyre );
    < Gjør POSTFIKS-operasjonene >
  }
}
```

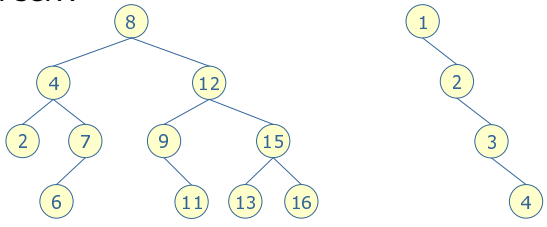


*Introduksjon til binære søketrær...*





- Binære søketrær er variant av binære trær hvor følgende holder for hver node i treet:
  - Alle verdiene i nodens **venstre** subtre er **mindre** enn verdien i noden selv.
  - Alle verdiene i nodens **høyre** subtre er **større** enn verdien i noden selv.




- Hva hvis vi har like verdier?
  (! Ukeoppgave !)

## NESTE GANG – *Uke 5, Forelesning 1 (W5.L1)*

- Neste gang, dvs. uke 5, forelesning 1 (W5.L1):
  Mer binære søketrær...

- Det vi skal se på er spesielt :
  - Implementasjon
  - Rekursive/ikke-rekursive søkemetoder i binære søketrær
  - Insetting, sletting som vanlig
  - Tidsforbruk i søking, insetting og sletting

- Husk innlevering av obligatorisk oppgave #1 før midnatt mandag 15. September 2003!

  OBS! TIL GRUPPELÆREREN!