UNIVERSITETET
I OSLO

INF2220: Forelesning 1

Praktisk informasjon

Analyse av algoritmer (kapittel 2)

(Binær)trær (kapittel 4.1-4.3 + 4.6)

PRAKTISK INFORMASJON

UNIVERSITETET I OSLO

Praktisk informasjon

- Kursansvarlige
  - Ragnhild Kobro Runde (ragnhilk@ifi.uio.no)
  - Ingrid Chieh Yu (ingridcy@ifi.uio.no)
  - Arne Maus (arnem@ifi.uio.no)

- Forelesninger
  - Torsdag 12-14, Simula OJD
  - Vær oppmerksom på at
    - ikke alt som er pensum, foreleses
    - ikke alt som foreleses står på lysarkene
    - alt som foreleses er (også) pensum

- Gruppetimer
  - 2 timer/uke seminarrom
  - 2 timer/uke terminalstue

Praktisk informasjon (forts)

- Obligatoriske oppgaver (tentative datoer)
  - Oblig 1 tirsdag 12/9
  - Oblig 2 tirsdag 3/10
  - Oblig 3 tirsdag 24/10
  - Oblig 4 tirsdag 7/11

- Eksamen (digital)
  - Mandag 4/12 kl 14.30-18.30(?)
  - Alle trykte og skrevne hjelpemidler tillatt

- Lærebok
  - Mark Allen Weiss: Data Structures and Algorithm Analysis in Java

INF2220 – algoritmer og datastrukturer

- Et av de mest sentrale grunnkursene i informatikk – og et av de vanskeligste!
- Kurset hever programmering fra et håndverk til en vitenskap.
- Eksamen krever både teoretiske og praktiske ferdigheter.
- Forelesninger og gruppetimer utfyller hverandre.
  - Forelesninger fokuserer på teori og ideer
  - Gruppetimene gir trening både i teorioppgaver og praktisk programmering
- Et arbeidskrevende modningsfag. Jobb med oppgaver gjennom hele semesteret!
- Sjekk hjemmesiden regelmessig for viktige beskjeder. Forelesningsplanen kan bli endret underveis.
- Kurset forutsetter INF1010, spesielt lister og rekursjon.

Hva er målet ditt med INF2220?

Hvordan skal du nå målet ditt?

TRÆR

Trær

Ulike typer trær:

- Generelle trær (kap. 4.1)
- Binærtrær (kap. 4.2 og 4.6)
- Binære søketrær (kap. 4.3)
- B-trær (kap. 4.7)

Aktuelle temaer:

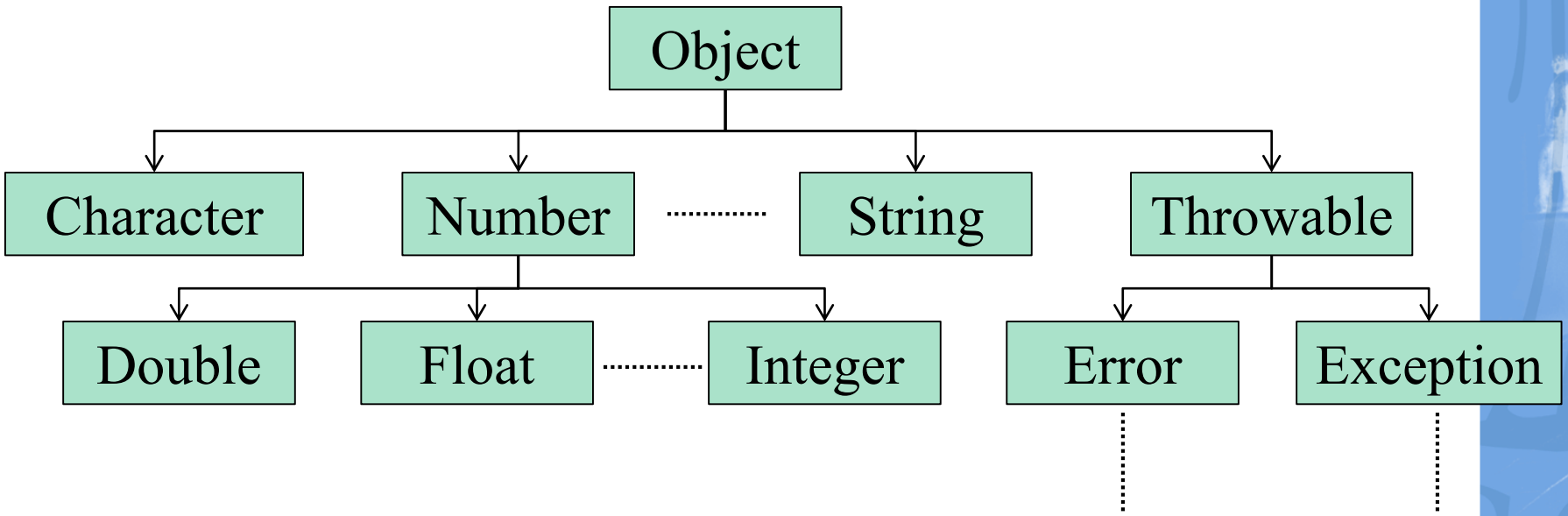
- Bruksområder
- Implementasjon
- Innsetting og fjerning av elementer
- Søking etter elementer
- Traversering

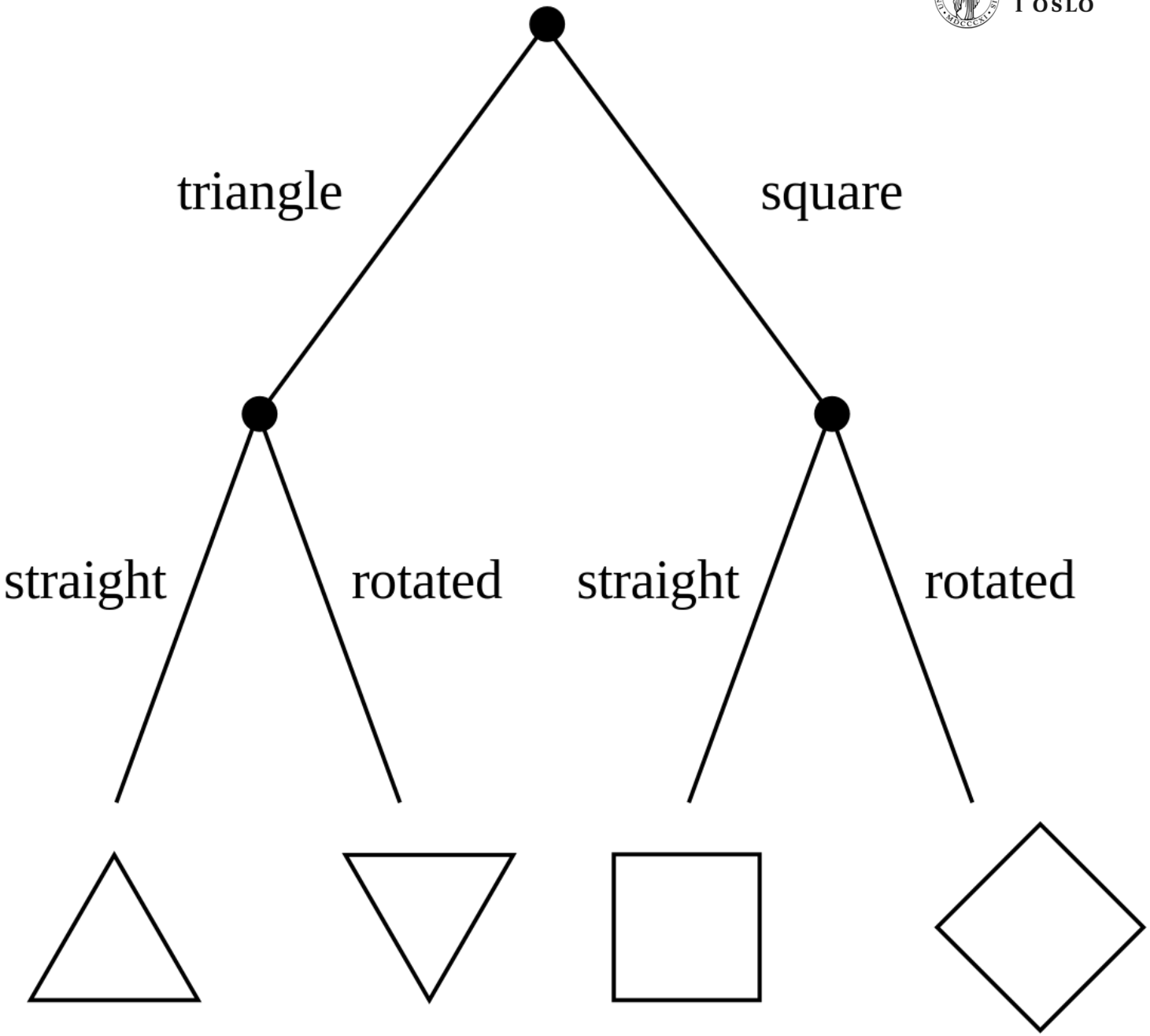
Generelle trær

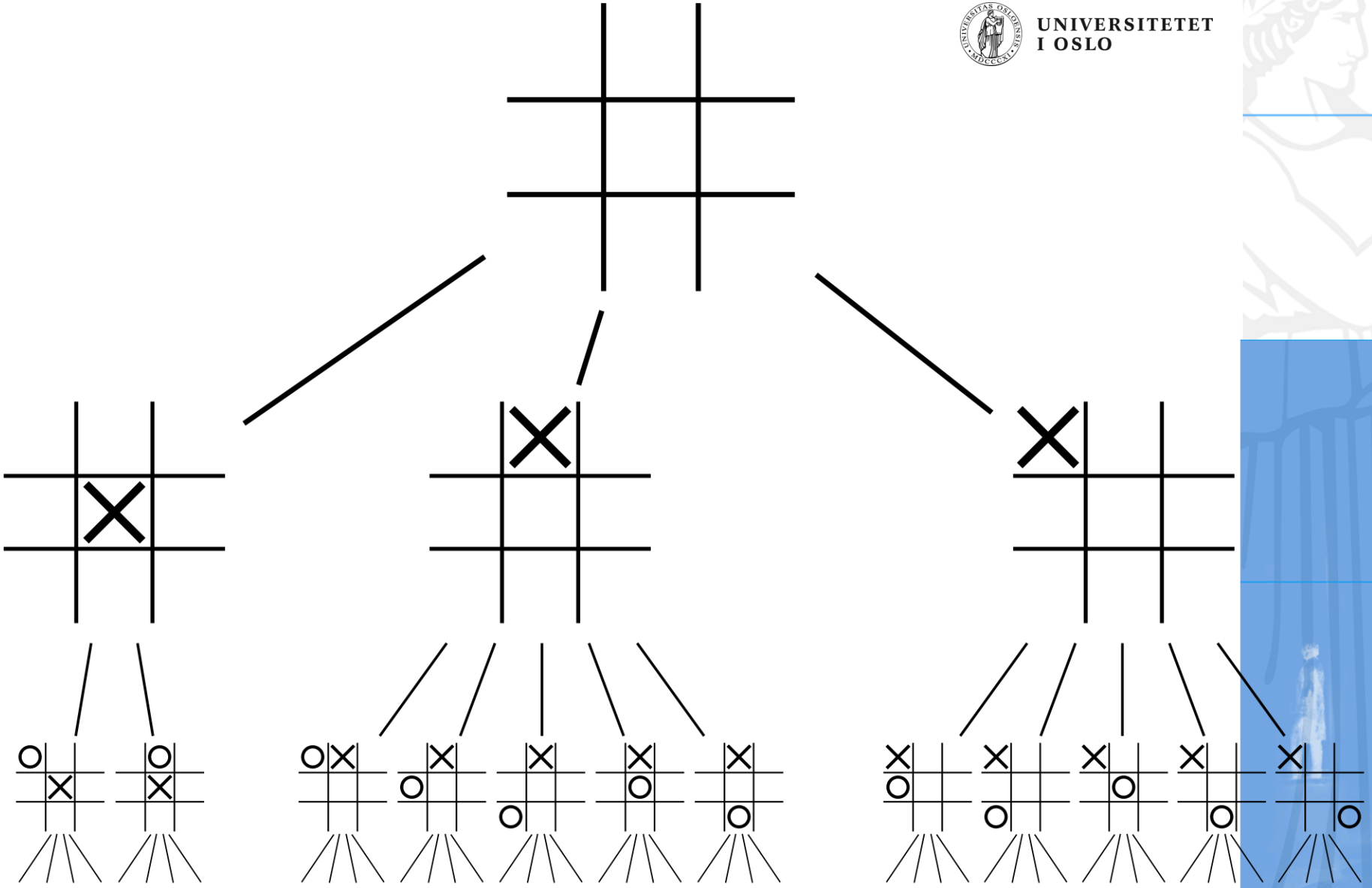
Trær brukes gjerne for å representere data som er organisert hierarkisk.

Typiske eksempler er:

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




triangle
square
straight
rotated
straight
rotated



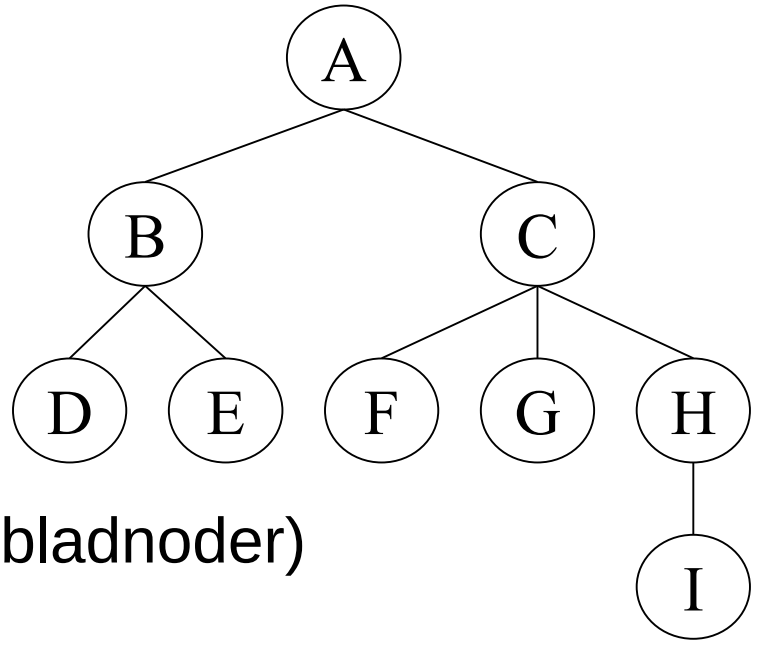
Terminologi

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

(Denne definisjonen er rekursiv!)

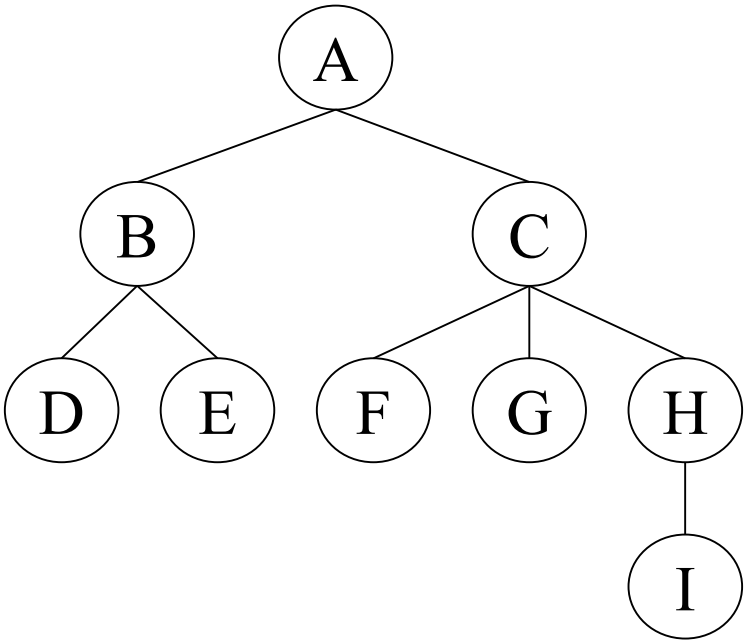
Eksempler:

- 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 løvnoder (bladnoder)
- A, B, C og H er indre noder



Diskuter

- Hvordan vil dere implementere en slik trestruktur i Java? (Tenk noder og pekere.)



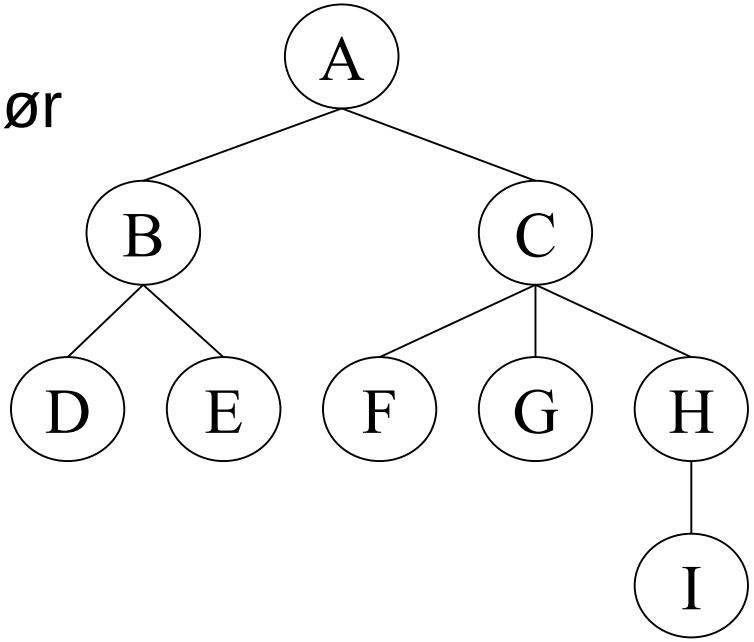
Traversering

UNIVERSITETET I OSLO

Å traversere et tre vil si å besøke alle (eller noen av) nodene i treet, for eksempel for å

- Lete etter en bestemt node (element).
- Sette inn en ny node (med et nytt element).
- Fjerne en node/element.
- Gjøre en beregning (eller utskrift) på nodene i treet.

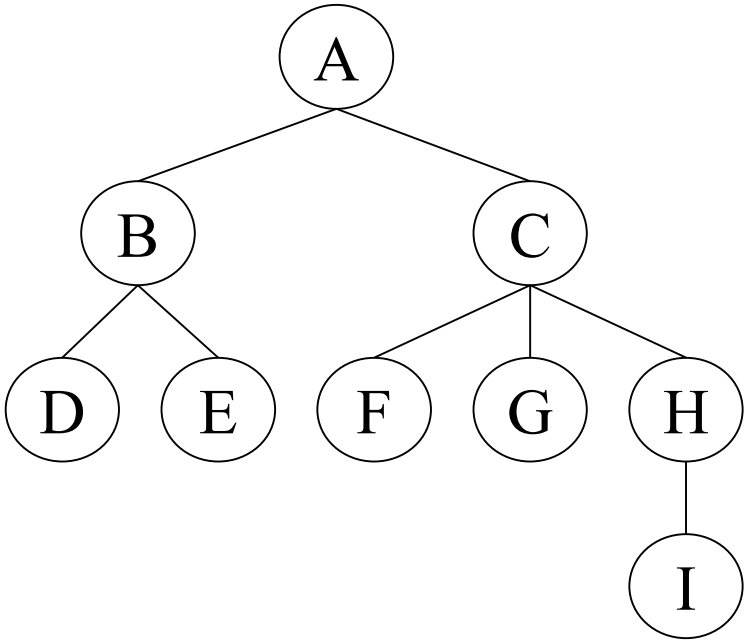
Problemet som skal løses avgjør traverserings-rekkefølgen.



Traversering (forts)

To populære traverseringsmåter:

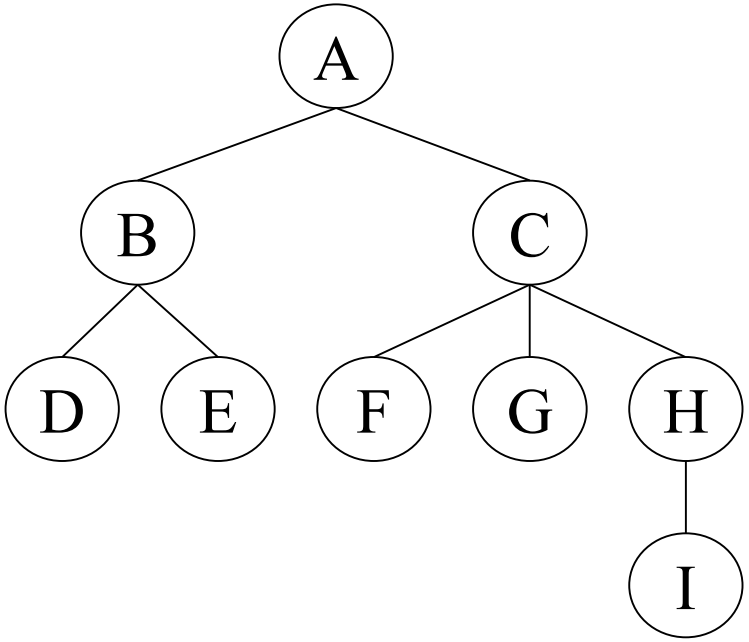
- Prefiks (preorder): behandle noden før vi går videre til barna.
- Postfiks (postorder): behandle noden etter at vi har besøkt alle barna til noden.



Mer terminologi

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.



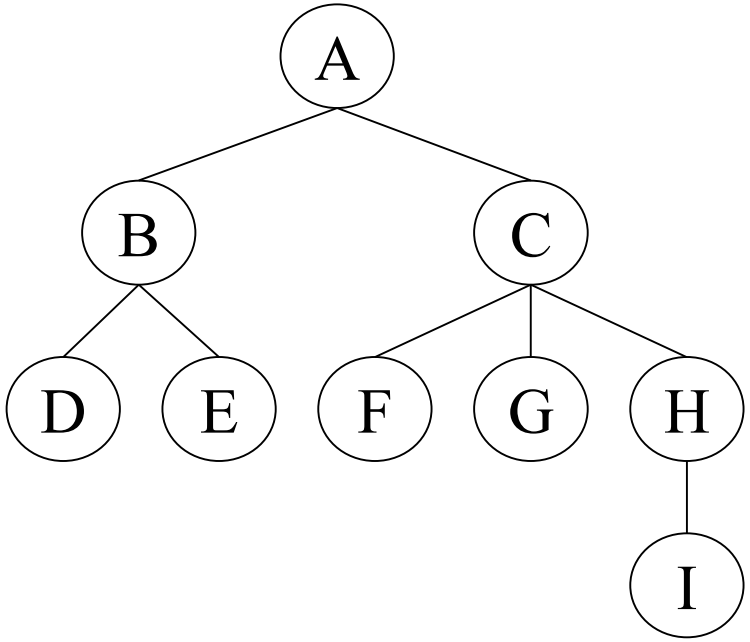
Dybde

Dybden til en node er definert 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);

void beregnDybde(int d) {
  this.dybde = d;
  for (<hvert barn b>) {
    b.beregnDybde(d+1);
  }
}
```



Høyde

Høyden til en node er definert som lengden av den lengste veien fra noden til en løvnode.

Alle løvnoder har dermed høyde 0.

Høyden til et tre er lik høyden til roten.

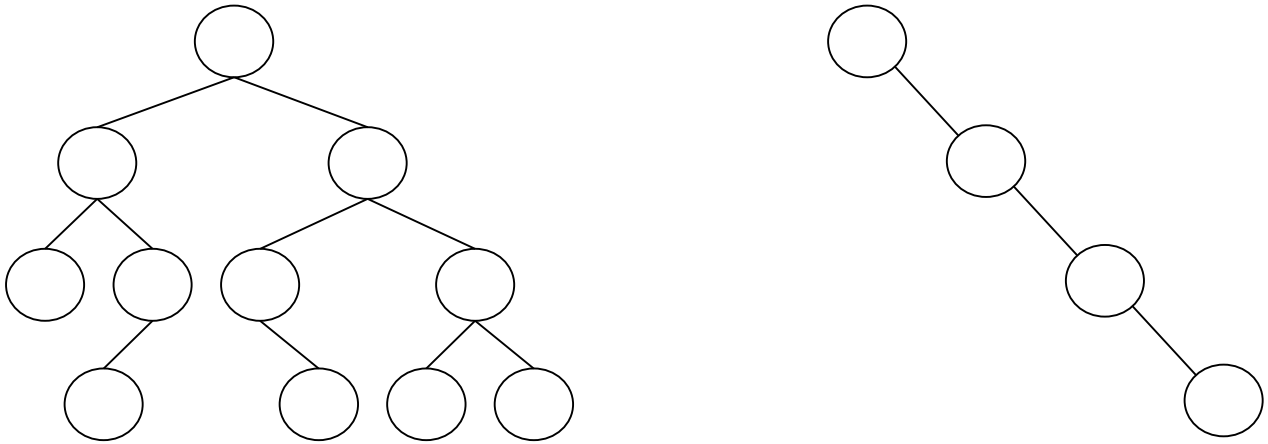
```
    // Kall:rot.beregnHoyde();

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

BINÆRTRÆR

Binærtrær

Et binærtre er et tre der hver node aldri har mer enn to 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!

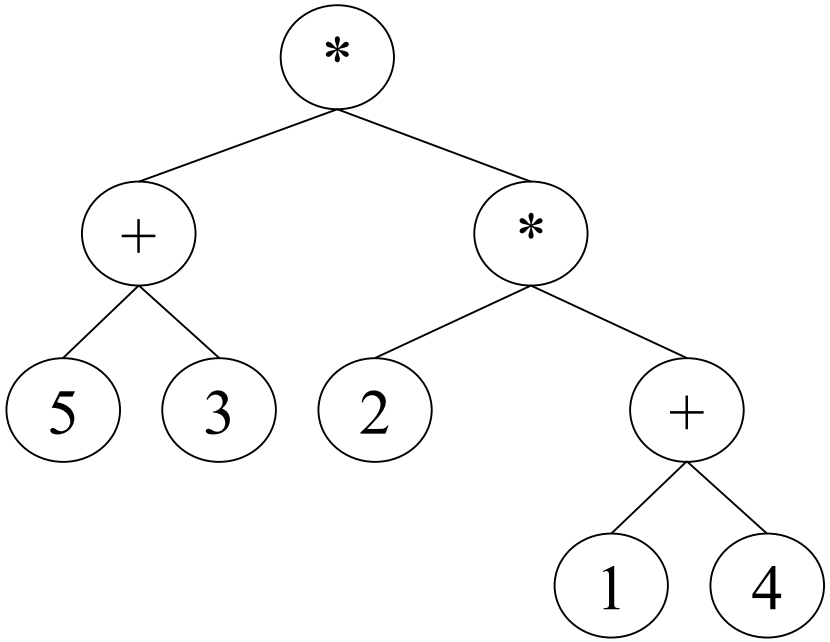
Implementasjon av binærtrær

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

```
class BinNode {
    Object element;
    BinNode venstre;
    BinNode hoyre;
}
```

Eksempel: Uttrykkstrær

I uttrykkstrær inneholder løvnodene operander (konstanter, variabler, …), mens de indre nodene inneholder operatorer.



Postfiks:

Infiks:

Prefiks:

Traversering av binærtrær - oppsummering

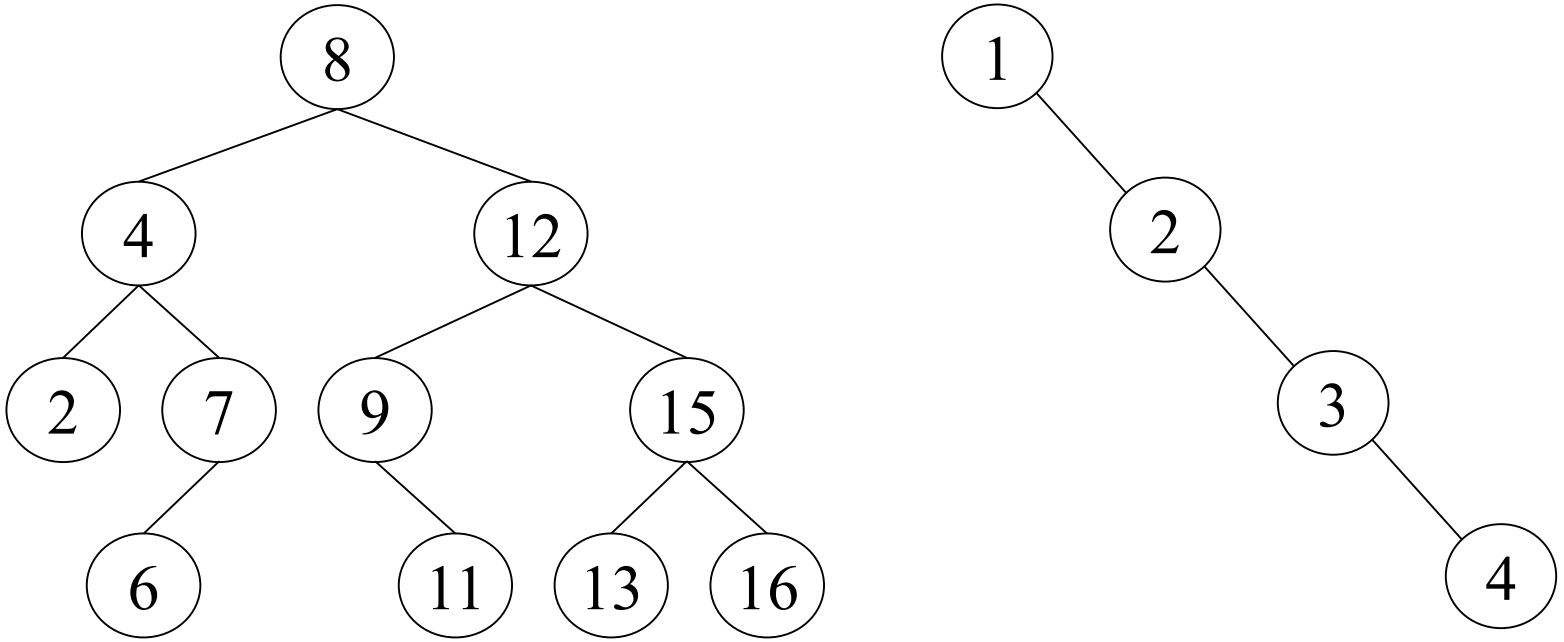
```
void traverser(BinNode n) {
    if (n != null) {
    < Gjør PREFIKS-operasjonene >
    traverser(n.venstre);
    < Gjør INFIKS-operasjonene >
    traverser (n.hoyre);
    < Gjør POSTFIKS-operasjonene >
}
```

BINÆRE SØKETRÆR

Binære søketrær

Binære søketrær er en variant av binærtrær hvor følgende for hver node i treet:

- Alle verdiene i venstre subtre er mindre enn verdien i noden selv.
- Alle verdiene i høyre subtre er større enn verdien i noden selv.



Hvordan håndtere noder med like verdier? (Ukeoppgave.)

Søking: Rekursiv metode

```
public BinNode finn(Comparable x, BinNode n) {
    if (n == null) {
        return null;
    } else if (x.compareTo(n.element) < 0) {
        return finn(x, n.venstre);
    } else if (x.compareTo(n.element) > 0) {
        return finn(x, n.hoyre);
    } else {
        return n;
    }
}
```

- Antagelse: Alle nodene i treet er forskjellige.
- Det må være mulig å sammenligne verdiene i treet – her er det brukt Java-interfacet Comparable.
- Her returneres noden som inneholder elementet. Ofte returneres elementet selv.

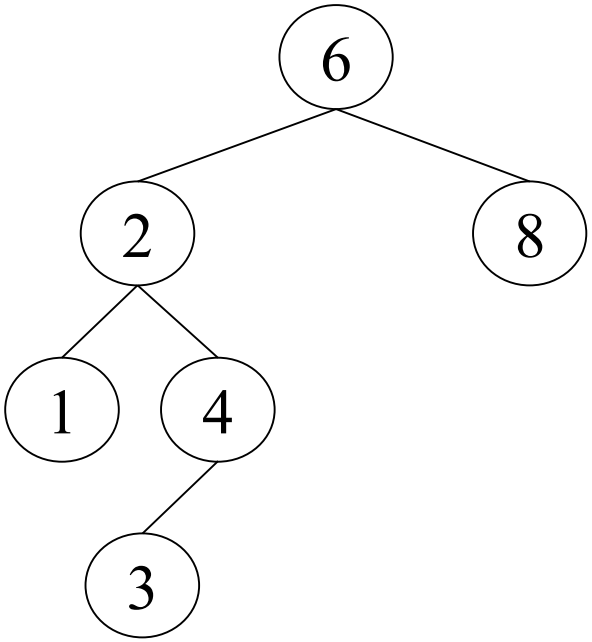
Søking: Ikke-rekursiv metode

```
public BinNode finn(Comparable x, BinNode n) {
    BinNode t = n;
    while (t != null && x.compareTo(t.element) != 0) {
        if (x.compareTo(t.element) < 0) {
            t = t.venstre;
        } else {
            t = t.hoyre;
        }
    }
    return t;
}
```

Innsetting

Ideen er enkel:

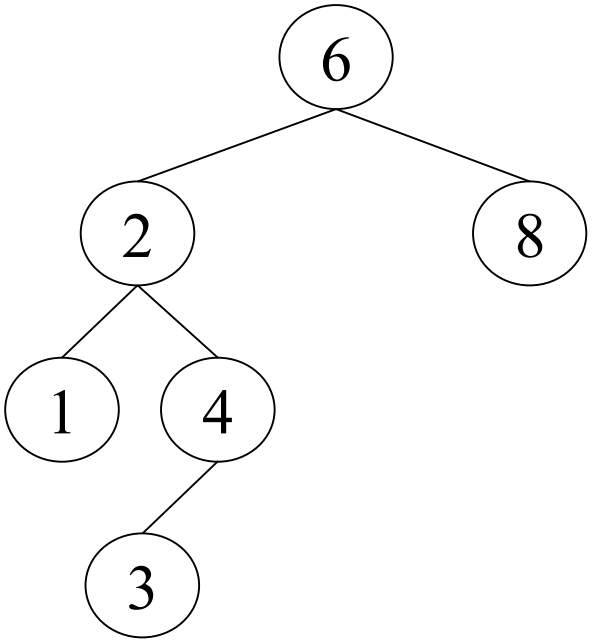
- Gå nedover i treet på samme måte som ved søking.
- Hvis elementet finnes i treet allerede gjøres ingenting.
- Hvis du kommer til en null-peker uten å ha funnet elementet: sett inn en ny node (med elementet) på dette stedet.



Sletting

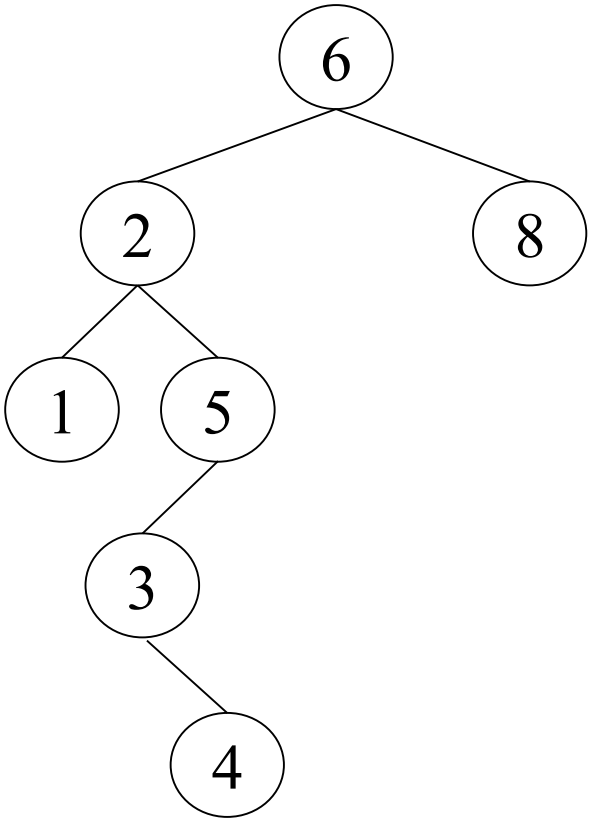
Sletting er vanskeligere. Etter å ha funnet noden som skal fjernes er det flere mulige situasjoner:

- Noden er en løvnode:
  - Kan fjernes direkte.
- Noden har bare ett barn:
  - Foreldrenoden kan enkelt hoppe over den som skal fjernes.



Sletting (forts)

- Noden har to barn:
  - Erstatt verdien i noden med den minste verdien i høyre subtre.
  - Slett noden som denne minste verdien var i.



ANALYSE AV ALGORITMER

Analyse av tidsforbruk

Hvor mye øker kjøretiden når vi øker størrelsen på input?

To typer analyse:

- Gjennomsnittlig tidsforbruk (average-case)
- Verste tilfelle (worst-case)

Alternative metoder:

- Implementere algoritmen og ta tiden for ulike typer og størrelser på input.
- Finne en enkel funksjon som vokser ”på samme måte” som kjøretiden til programmet.

O-notasjon

Generelt er vi ikke interessert i nøyaktig hvor mye tid et program bruker, men vil heller prøve å angi i hvilken **størrelsesorden** løsningen ligger.

Definisjon

La T(n) være kjøretiden til programmet.

T(n) = O(f(n)) hvis det finnes positive konstanter c og n_0 slik at

$$T(n) \leq c * f(n) \qquad \text{når } n > n_0.$$

O(f(n)) er da en øvre grense for kjøretiden.

Oppgaven er å finne en f(n) som er minst mulig.

Vanlige funksjoner for O()

Funksjon	Navn
1	Konstant
log n	Logaritmisk
n	Lineær
n log n	
n^2	Kvadratisk
n^3	Kubisk
2^n	Eksponensiell
n!	

O-notasjon er en veldig forenklet (asymptotisk) måte å angi tidsforbruk på, og vi forkorter så mye som mulig. Merk at konstanter allerede ligger i definisjonen.

Eksempler:

- n/2, n, 2n: O(n)
- $n^2 + n + 1$: $O(n^2)$
- $\log_2 n$, $\log_{10} n$: O(log n)

Neste forelesning: 31. august

BALANSERTE SØKETRÆR