Uke 1, Forelesning 2

- Oversikt
- Essensen i faget: Effektive algoritmer
- Matematiske forutsetninger
  - Logaritmer, regneregler for logaritmer
  - Eksponenter, regneregler for eksponenter
  - Rekker og summer
  - Bevis
    - Induksjonsbevis
    - Motbevis ved hjelp av moteksempel
    - Bevis ved hjelp av selvmotsigelse
  - Noe terminologi
    - Pseudo-kode
    - Måle reel tid eller teoretisk beregne/estimere tid

OVERSIKT – *Uke 1, Forelesning 2 (W1.L2)*

- TEMA #1:
  Introduksjon til METODEKALL og REKURSJON
- TEMA #2:
  Introduksjon til KOMBINATORISKE SØK

- Introduksjon til OBLIG 1:
  DRONNING OPPGAVEN

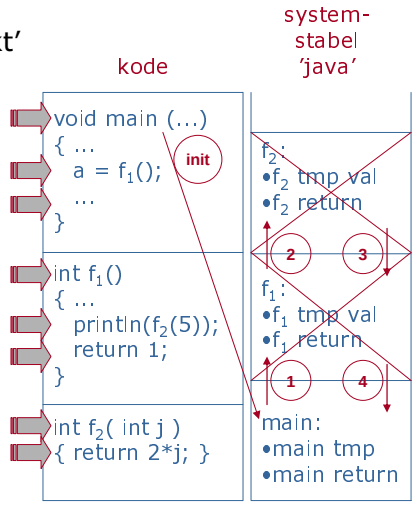
TEMA #1

© M. C. Escher

Metodekall og Rekursjon

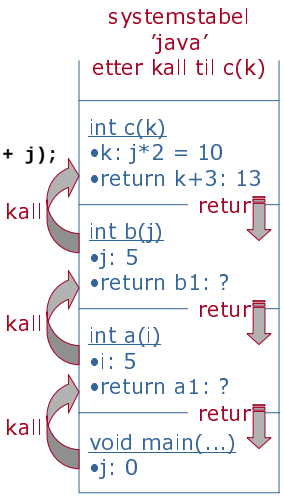
METODEKALL – *Definisjon*

- Når en metode $f_2( )$ kalles:
  - Det opprettes et 'dataobjekt' for $f_2( )$ som legges på toppen av en stabel i kjøresystemet ('java') i hovedhukommelsen.
  - Det som tidligere lå på toppen av stabelen (stack'en) var dataobjektet for den metoden som kalte $f_2( )$, dvs. $f_1( )$.
  - Når $f_2( )$ er ferdig (dvs. returnerer), tas $f_2( )$'s dataobjekt av stabelen.
  - I bunn av stabelen ligger 'dataobjektet' til 'main( )'.

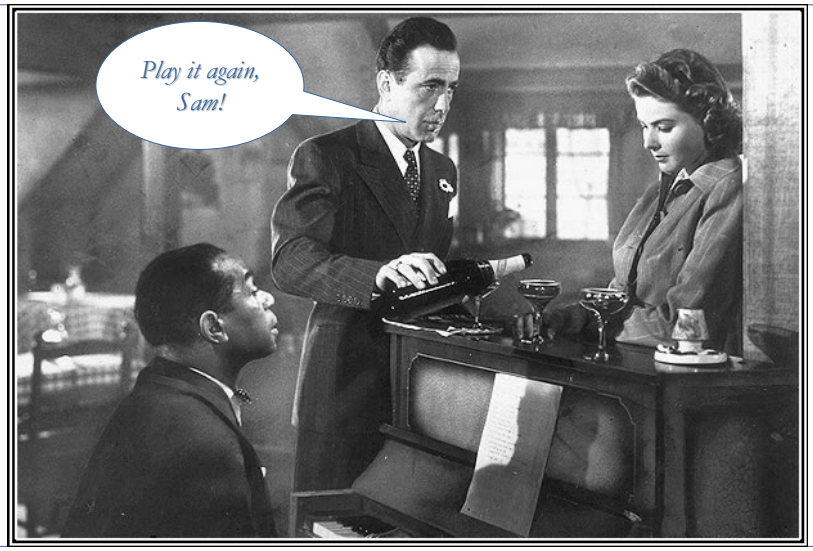


METODEKALL – *Eksempel*

```
import java.io.*;

public class MetodeTest
// Viser bruk av stacken i Javas kjøresystem (java)
{  public static void  main (String[] args)
      {  int j =0;
         MetodeTest t = new MetodeTest();
         j= t.a(5);
         System.out.println("Kall a(5)->b->c gir: " + j);
      }

      int a(int i)
      {  int a1 = b(i)+1;
         return a1;
      }

      int b(int j)
      {  int b1 = c(j*2)+2;
         return b1;
      }

      int c(int k)
      {  return k +3;
      }
}
```



REKURSJON



REKURSJON – *Definisjon*

- Rekursjon:

  Når en metode (funksjon, prosedyre, subrutine osv.) kaller seg selv!

- Prinsipp:

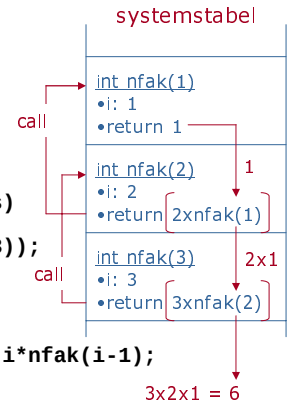
NB!
Skal helst terminere, ikke fortsette å kalle seg selv i det uendelige!

```
int funksjon(verdi)
{
    if ( kan-terminere )
       return en-siste-verdi;
    else
       return funksjon(en-verdi-nærmere-terminering);
}
```

- Eksempel: N-fakultet = N! = 1 x 2 x 3 x ... x (N-1) x N

  (Engelsk for fakultet: Factorial)

```
import java.io.*;

public class FakTest
// Viser bruk rekursiv metode: nfak()
{
   public static void  main (String[] args)
   { FakTest f = new FakTest();
     System.out.println("3! = " + f.nfak(3));
   }

   int nfak(int i)
   {
     if (i  ==  1 ) return 1; else return i*nfak(i-1);
   }
}
```



- Husk å teste om det skal avsluttes, og helst først!
  - a) FEIL: Vil overfylle stabelen i kjøresystemet…

```
int nfak(int i)
{
    return i* nfak(i-1);
}
```

  - b) OK: Tester først.

```
int nfak(int i)
{
    if (i == 1 )
       return 1;
    else
       return i* nfak(i-1);
}
```

REKURSJON – *Bruksområder*

- Visse typer problemer er naturligere å løse med rekursjon
  - Eksempel:
    Skriv to prosedyrer – en rekursiv og en ikke-rekursiv – for å beregne summen av N første heltall,
    dvs. 1 + 2 + 3 + … + N.
  - Vilkårlige tallrekker (særlige samme operasjon gjentatt på hele rekken) ofte er slike problemer som lar seg løse mer naturlig med rekursjon.
- Det kan hende at rekursive løsninger ikke alltid er ønskelige, men rekursjon er uansett en effektiv tenkemåte når en skal designe algoritmer.

TEMA #2

Kombinasjoner
og
Kombinatoriske Søk

© M. C. Escher

OM – *Kombinasjoner*

- Noen problemer er slik at vi må lete gjennom alle mulige kombinasjoner for så å plukke ut den/de kombinasjoner som tilfredsstiller våre krav (dvs. de som er løsninger på problemet).
- To eksempler:
  - Finn alle virkelig forskjellige måter å plassere N dronninger på et sjakkbrett slik at ingen av dronningene slår hverandre.
  - Finn korteste rundtur blant M byer hvor du besøker hver by bare en gang (gitt at du vet alle avstander fra hver by til alle de andre).
- Vi har da problemer der utfordringen er i to trinn:
  - Greie å lage alle interessante kombinasjoner
  - Teste om en kombinasjon er en løsning for vårt problem.
- N.B : Ofte blander vi inn selve problemet i 'bevisst' genereringen av alle mulige kombinasjoner, slik at vi bare genererer kombinasjoner det er 'håp for'.

OM – *Å Lage Kombinasjoner*

- Greie å lage alle interessante kombinasjoner av n personer/byer/dronninger/gjenstander…
- Spiller ingen rolle hva. Dermed kan enkelt representeres med heltallene 0, 1, 2,..., N-1.
- To eksempel problemer:
  - Telle gjennom alle N-sifrete tall og bare bruke de tall hvor alle sifrene er ulike.
    Kan representere for eksempel følgende problem: Hvor mange besøksrekkefølger finnes det hvis en skal besøke alle byer med start i en vilkårlig by og uten å besøke samme by to ganger?
  - Generere tall (gjerne i stigende rekkefølge) hvor i utgangspunktet alle sifrene er ulike. Finn på noe en slik samling av tall kan representere!

- Vi må generere 'alle' mulige kombinasjoner for så å teste om en kombinasjon er en løsning for vårt problem.

  Eller må vi?

- Å bare generer de kombinasjonene det er 'håp for' kalles avskjæring

- Eksempel:
  Lag alle kombinasjoner av rekkefølger en skoleklasse på 9 elever kan gå inn i klasserommet på, hvor Per (elev 4) og Pål (elev 7) **ikke** må gå etter hverandre (de er uvenner).
  - Håpløse tilfeller: Alle 47??????, ?47??????, ??47?????...
  - Avskjæring: Gitt du har hatt 4, forkast alle som begynner med 7 og generer de andre.

- **Spørsmål**: Hvor mange rekkefølger kan N ting (nummerert 0, 1, …, N-1) stilles opp i uten at tingen (tallet) gjentas (dvs. uten at samme ting/tall forekommer mer enn én gang i rekken)?

- **Svar** : N! = 1 × 2 × … × N ulike rekkefølger.
  Kalles **N-fakultet = N!** og gir alle mulige **permutasjoner** av N gjenstander. NB! Permutasjoner er kombinasjoner der rekkefølgen teller, dvs. der rekken 1, 2, 3 ≠ rekken 3, 2, 1.

- Eksempel: For N = 3 er N! = 1 × 2 × 3 = 6, og gir…

  0 1 2 0 2 1 1 0 2 1 2 0 2 0 1 2 1 0

  Men … hvordan har vi formulert at det blir N-fakultet ???

OM – *Å Formulere Kombinatoriske Løsninger*

- Tenk på det slik: Du har N tomme bokser (eller plasser) ved siden av hverandre og N fotballer nummerert fra 1 til N.

- Når tar vi 1 ball av N baller og putter den i den første boksen eller plassen. Vi kunne valgt ball nr. 1, 2, 3 og opp til N.
- Så putter vi en ball i neste boks. Men da har vi kun N-1 valg fordi 1 av de N er valgt og er i første boks allerede!
- Våre valgmuligheter i de to første boksene er altså slik at vi har N muligheter i den første boksen, og for hver ball i første boksen kan vi velge 1 blant N-1 resterende baller:
  Altså har vi N × (N-1) mulige kombinasjoner i de to første boksene…
- Med samme logikk, har vi N × (N-1) × (N-2) × … × 3 × 2 × 1 mulige kombinasjoner, som gir oss N!
- Se nå på rekkefølgen vi hadde generert for N = 3 og N! = 6:

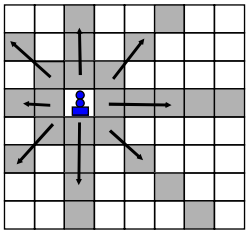
  0 1 2 0 2 1 1 0 2 1 2 0 2 0 1 2 1 0

NESTE GANG – *Generering av Permutasjoner*

- Hvordan generere alle permutasjoner (ombyttinger eller rekkefølger) av 0, 1, 2, ..., (N-1) i et program?

OBLIG 1 – *Introduksjon*

- En Dronning i sjakk kan slå alle brikker som står på…
  samme rad,
  samme kolonne,
  eller samme diagonaler (dvs. rette skrålinjer)…

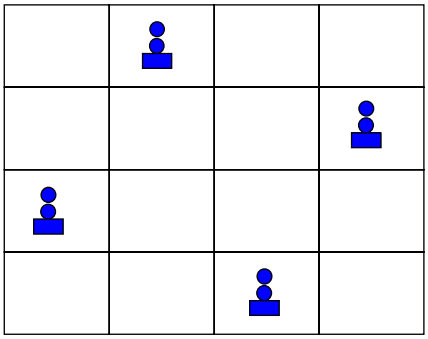


OBLIG 1 – *Oppgaven*

- Bruk permutasjoner til å skrive ut alle interessante løsninger av 'n' dronninger på et N x N sjakkbrett hvor ingen dronning kan slå en annen.
- Ser at N=1 har 1 løsning, mens N = 2 og 3 ikke har løsning. Hva med N=4 ?
  Løs oppgaven generelt, og test med N=8.
- Hvorfor permutasjoner:
  - La p[i] si hvilken kolonnenummer. Vi vil plassere en dronning i rad nr 'i'. Da slipper vi å generere mange håpløse kombinasjoner…
- Vi er ikke interessert i 'nye' løsninger som kan lages fra en allerede presentert løsning ved:
  - å vippe sjakkbrettet rundt om en midtlinje eller diagonal
  - rotere brettet 90, 180 eller 270 grader

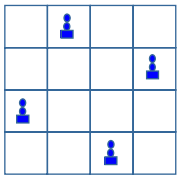
OBLIG 1 – *En løsning for N = 4*

- Her er en løsning for N = 4:

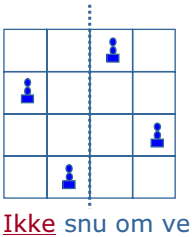


OBLIG 1 – *Regler om Løsningen*

- Husk å fjerne de løsninger som bare er en snuing av brettet om horisontal/vertikal midtlinje eller rotering av brettet som i eksempelet nedenfor for N = 4…



Utgangsløsning



Ikke snu om vertikal midtlinje!

OBLIG 1 – *Konkurransen…*

- **NB!** Konkurranse…
- **Tre raskeste riktige og velstrukturerte** løsninger for N = 10, som bare skriver ut de ulike innholdene av p[ ] for de forskjellige løsningene…
  - Alle tre får ”anerkjennelse” … skriftlig!
  - Første får i tillegg … noe ganske godt ☺
  - Nummer to og tre får i tillegg en pose ”Twist” hver…
  - Uke 5, mandag 15. september 2003, skal de tre presentere (forklare) sine løsninger og kjøre de på auditoriet…

TILLEGG – *Notater fra Forelesningen #1*

- **NB!** Disse er tilleggsnotater fra forelesningen, lagt ved etter forelesningen…

Hale rekursjon:

```
void rfunk( ... )
{  // Gjør noe først...
   ...
   ... rfunk( ... );
}
```

Test, kall rekursivt og return til slutt.

Hode rekursjon:

```
void rfunk( ... )
{  ... rfunk( ... );
   ...
   // Gjør noe sist...
}
```

Test og kall rekursivt, ...

og return etter du har gjort noe til slutt.

- **NB!** Disse er tilleggsnotater fra forelesningen, lagt ved **etter** forelesningen…
- **Husk** at **permutasjoner** er kombinasjoner der **rekkefølgen er viktig**. Eks.: rekken 1, 2, 3 ≠ rekken 3, 2, 1. som er to permutasjoner av tallmengden {1, 2, 3}.
- En annen definisjon er at permutasjoner er **gjenordninger** av en rekke.

  Eksempelvis vil mengden {1, 2, 3} kan gjenordnes for å gi 3! = 6 rekker med distinkte ordninger (rekkefølger):

  [1, 2, 3] [1, 3, 2] [2, 1, 3] [2, 3, 1] [3, 1, 2] [3, 2, 1]