Uke 8,
Forelesning 2

- Forutsetninger for og essensen i faget
- Metodekall, rekursjon, permutasjoner
- Analyse av algoritmer
- Introduksjon til ADT'er
- De første ADT'er: Lister, stabler og køer

Uke 1, Uke 2 og Uke 3

- Flere ADT'er: Generelle trær, binære trær og binære søketrær

Uke 4 og Uke 5

- Flere ADT'er: Hashing, hash-tabeller
- ADT'er for disk-datastrukturer introduseres: Utvidbar hashing, B-Trær
- Prioritetskøer & Heap-implemantasjonen

Uke 6 og Uke 7

- Disjunkte mengder

Uke 8.1

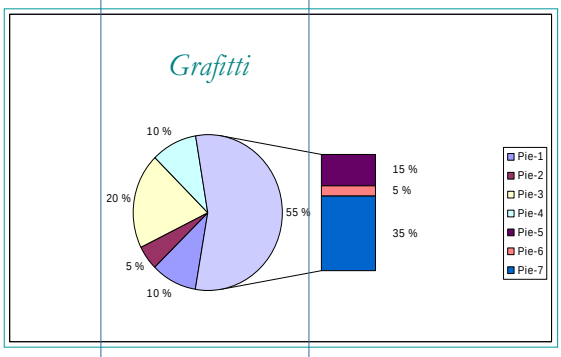
OVERSIKT – *Uke 8, Forelesning 2 (W8.L2)*

Vi introduserer ADT'en for grafer.

Spesifikt skal vi se på:

- Definisjon av en graf (kapittel 9.1)
- Varianter av grafer
- Intern representasjon av grafer (kapittel 9.1.1)
- Topologisk sortering (kapittel 9.2)

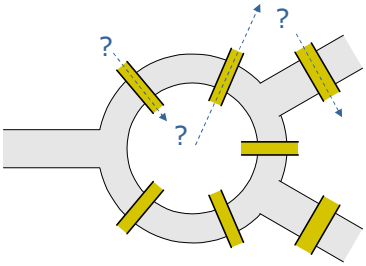
TEMA: GRAFER



Introduksjon til Grafer

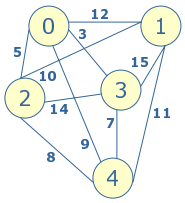
Det første grafteoretiske problemet:
Broene i Königsberg

- Er det mulig å ta en spasertur som krysser hver av broene nøyaktig en gang?
- Dette problemet ble løst av Euler allerede i 1736!

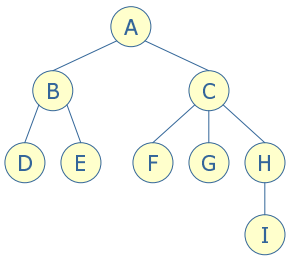


Mange grafer har vi sett allerede!

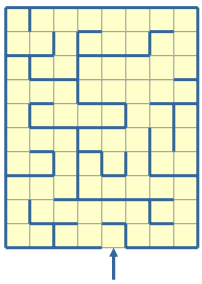
Rundresie



Trær

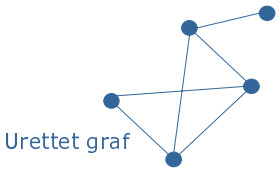


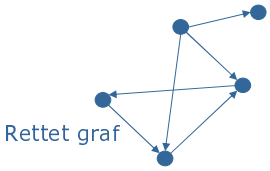
Labyrint



Hva er en graf?

- En graf G =(V,E) består av en mengde noder, V (Eng: 1 vertex, mange vertices), og en mengde kanter, E (Eng: edges)
- $|V|$ og $|E|$ er henholdsvis antall noder og antall kanter i grafen
- Hver kant er et par av noder, dvs. (u, v) slik at $u, v \in V$
- En kant (u, v) modellerer at u er relatert til v
- Dersom nodeparet i kanten (u, v) er ordnet (dvs. at rekkefølgen har betydning), sier vi at grafen er rettet, i motsatt fall er den urettet

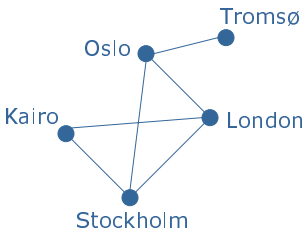




- En graf er den mest fleksible datastrukturen vi kjenner. Så-å-si "alt" kan modelleres ved hjelp av grafer.

Hvorfor grafer?

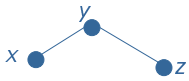
- De dukker opp i veldig mange problemer i hverdagslivet:
  - Flyplassystemer
  - Datanettverk
  - Trafikkflyt
  - Ruteplanlegging
  - VLSI (chip design)
  - og mange flere . . .



- Grafalgoritmer viser veldig godt hvor viktig valg av datastruktur er mhp. tidsforbruk
- Det finnes graf-baserte eller graf-relaterte grunnleggende algoritmeteknikker som løser mange ikke-trivielle problemer raskt

Definisjoner og varianter av grafer

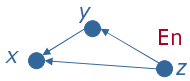
- Node *y* er nabo-node (eller etterfølger) til node *x* dersom $(x, y) \in E$



x og *y* er naboer, *y* og *z* er naboer, men *x* og *z* er ikke naboer.

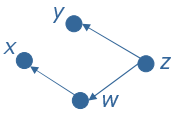
z er nabo til *y*, men *y* er ikke nabo til *z*.

- En graf er vektet dersom hver kant har en tredje komponent, kalt kostnad eller vekt



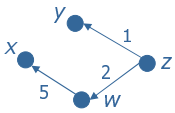
En vektet og rettet graf.

- En vei (eller sti) i en graf er en sekvens av noder $v_1, v_2, v_3, \ldots, v_n$ slik at $(v_i, v_{i+1}) \in E$ for $1 \le i \le n-1$
- Lengden til veien er lik antall kanter på veien, dvs. $n - 1$



<z, w, x> er en vei av lengde 2

- Kosten til en vei er summene av vektene langs veien

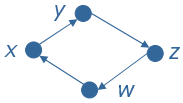


<z, w, x> er en vei med kost 7

- En vei er enkel dersom alle nodene (untatt muligens første og siste) på veien er forskjellige
- Våre grafer har vanligvis ikke "loops", (v, v), eller "multikanter" (to like kanter):

loop multikant vanlig rettet graf

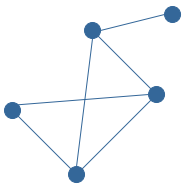
- En løkke (syklus) i en rettet graf er en vei med lengde ≥ 1 slik at $v_1 = v_n$. Løkken er enkel dersom stien er enkel
- I en urettet graf må også alle kanter i løkken være forskjellige



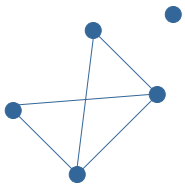
<x, y, z, w, x> er en enkel løkke

GRAFER – *Grafvarianter og definisjoner #3*

- En rettet graf er asyklisk dersom den ikke har noen løkker
- En rettet, asyklisk graf blir ofte kalt en DAG (Directed, Acyclic Graph)
- En urettet graf er sammenhengende dersom det er en vei fra hver node til alle andre noder
  (og ikke-sammenhengende ellers)



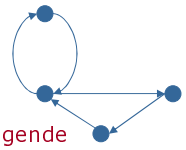
sammenhengende



ikke sammenhengende

GRAFER – *Grafvarianter og definisjoner #4*

- En rettet graf er sterkt sammenhengende dersom det er en vei fra hver node til alle andre noder
- En rettet graf er svakt sammenhengende dersom den underliggende urettede grafen er sammenhengende



sterk sammenhengende

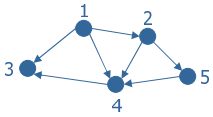
svak sammenhengende

- Graden til en node i en urettet graf er antall kanter mot noden
- Inngraden til en node i en rettet graf er antall kanter inn til noden
- Utgraden til en node i en rettet graf er antall kanter ut fra noden.

Grad(v) = 3

Inngrad(v) = 1
Inngrad(v) = 4

Hvordan representere grafer?

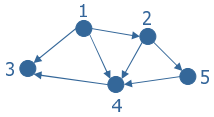


Nabo-matrise

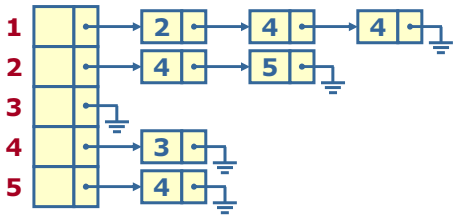
	1	2	3	4	5
1	0	1	1	1	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	1	0	0
5	0	0	0	1	0

- Bra hvis "tett" graf, dvs. $|E| = \Theta(|V|^2)$
- Tar $O(|V|)$ tid å finne alle naboer

Alternativt



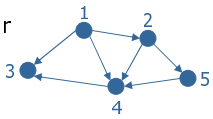
Nabo-liste



- Bra hvis "tynn" graf
- Tar $O(\text{Utgrad}(v))$ tid å finne alle naboer til v
- De fleste grafer i det virkelige liv er tynne!

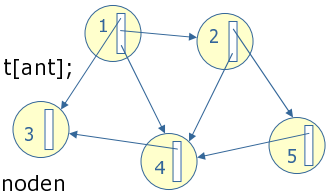
Objekter & array

- I Java kan grafer også representeres ved en kombinasjon av node-objekter og etterfølger arrayer
- Array-lengden kan være en parameter til node-klassen:




```
class Node
{  int antEtterf;
   Node[ ] etterf; Float[ ] vekt;

   Node(int ant)
   {  etterf = new Node[ant]; vekt = new Float[ant];
      antEtterf = ant;
   }
}
```



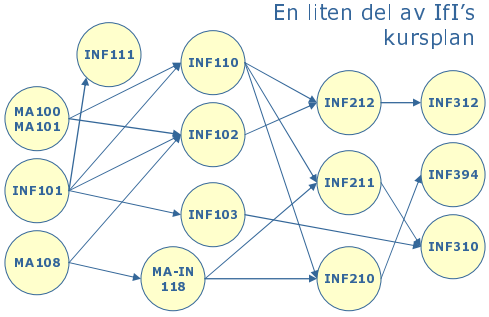
- Må da vite antall etterfølgere når vi genererer noden
- Eventuelt kan vi *estimere* en øvre grense og la siste del av array'en være tom
- Vi trenger da en variabel som sier hvor mange etterfølgere en node *faktisk* har

Introduksjon til topologisk sortering

- En topologisk sortering er en ordning (rekkefølge) av noder i en DAG slik at dersom det finnes en vei fra v_i til v_j, så kommer v_j **etter** v_i i ordningen
- Topologisk sortering er umulig hvis grafen har en løkke
- Vanligvis er det flere mulige løsninger

- Eksempel:
  forutsetter/bygger på graf

En liten del av IfI's kursplan

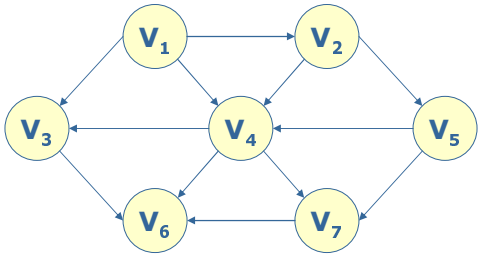


- En topologisk sortering er for eksempel en "lovlig" rekkefølge å ta alle kursene på IfI i

- Følgende enkle algoritmen finner en topologisk sortering (dersom det er noen):

- Finn en node med inngrad = 0
- Skriv ut noden, og fjern noden og utkantene fra grafen (marker noden som "ferdig" og reduser inngraden til nabonodene)
- Gå tilbake til punkt 1

Eksempel (figur 9.4)



MAW side 295, figur 9.5:

```
void topsort()
{  Node v;
   for (int teller = 0; teller < ANTALL_NODER; teller++)
   {  v = finnNyNodeMedInngradNull();
      if (v == null)
         error("Løkke funnet!");
      else
      {  < Skriv ut v som node 'teller' >
         for < hver nabo w til v >
         w.inngrad--;
      }
   }
}
```

- Denne algoritmen er $O(|V|^2)$ siden finnNyNodeMedInngradNull ser gjennom hele node/inngrad-tabellen hver gang
- Dette er unødvendig mye, siden bare noen få av verdiene kommer ned til 0 hver gang

En forbedring er å holde alle noder med inngrad=0 i en **boks**.

Boksen kan implementeres som en stakk eller en kø:

- Plasser alle nodene med inngrad=0 i boksen.
- Ta ut en node *v* fra boksen.
- Skriv ut *v*.
- (Fjern *v* fra grafen og) reduser inngraden til alle etterfølgerne.
- Dersom noen av etterfølgerne får inngrad=0, settes de inn i boksen.
- Gå tilbake til punkt 2.

MAW side 297, figur 9.7:

```
void topsort()
{  Kø k = new Kø();
   Node v;

   for < hver node >
      if ( v.inngrad == 0 ) k.settInn(v);

   while ( !k.isEmpty() )
   {  v = k.taUt();
      < Skriv ut v >
      for < hver nabo w til v >
      {  w.inngrad--;
         if (w.inngrad == 0) k.settInn(w);
      }
   }
}
```

- Forutsatt at vi bruker nabolister, er denne algoritmen $O(|E| + |V|)$.
- Kø/stakk-operasjoner tar konstant tid, og hver kant og hver node blir bare behandlet én gang.

NESTE GANG – *Oppsummering*

ALMIRA KARABEG foreleser!
Fortsetter med grafer, korteste vei, én-til-alle for:

- Uvektet graf (kapittel 9.3.1)
- Vektet rettet graf uten negative kanter (kapittel 9.3.2)
- Vektet rettet graf med negative kanter (kapittel 9.3.3)