F11 - Mängd, graf

5DV149 Datastrukturer och algoritmer
Kapitel 13.1–13.2, 17

Niclas Börlin
niclas.borlin@cs.umu.se

2020-02-24 Mon

Innehåll

- Mängd
- Graf
- OU 5

Mängd

Mängd

- Modell: En påse (men man kan inte ha två likadana element).
- Organisation:
  - En *oordnad* samling av element som är av samma typ.
  - Grundmängden behöver inte vara ändlig (ex. heltal) men dataobjekten är ändliga.
  - Kan inte innehålla två element med likadana värden.
  - En mängd kan inte innehålla mängder.
- En mängd som kan innehålla flera element med samma värde kallas multimängd (*multiset*) eller påse (*bag*).
  - Vid textsökning betraktas ofta ett dokument som en påse av ord (*bag of words*), där bara ordens frekvens räknas (jfr. google).

Specifikation (1)

```
abstract datatype  Set(val)
  Empty()                                    → Set(val)
  Single(v:val)                              → Set(val)
  Insert(v:val, s:Set(val))                  → Set(val)
  Union(s:Set(val), t:Set(val))              → Set(val)
  Intersection(s:Set(val),t:Set(val))        → Set(val)
  Difference(s:Set(val), t:Set(val))         → Set(val)
  Isempty(s:Set(val))                        → Bool
  Member-of(v:val, s:Set(val))               → Bool
  Choose(s:Set(val))                         → val
  Remove(v:val,s:Set(val))                   → Set(val)
  Equal(s:Set(val), t:Set(val))              → Bool
  Subset(s:Set(val), t:Set(val))             → Bool
```

Specifikation (2)

- Boken har tagit med de vanliga matematiska mängdoperationerna.
- Alla behövs inte.
- Följande operationer räcker:

```
abstract datatype  Set(val)
  Empty()                              → Set(val)
  Isempty(s:Set(val))                  → Bool
  Insert(v:val, s:Set(val))            → Set(val)
  Choose(s:Set(val))                   → val
  Remove(v:val,s:Set(val))             → Set(val)
```

Konstruktion av mängd, dubletter

- De flesta konstruktioner måste kunna hantera att det inte får finnas dubletter i en mängd.
- Mängd som lista har två alternativ:
  - Se till att listan inte har dubbletter (krav på `Insert` och `Union`).
  - Låt listan innehålla dubbletter (krav på `Equal`, `Remove`, `Intersection`, `Difference`).

Konstruktion av mängd som lista

- Komplexitet:
  - Metoder som kräver sökningar i listan: $O(n)$.
  - Binära mängdoperationerna mellan två listor med m och n element har komplexitet $O(mn)$.
- Listan kan konstrueras på olika sätt.
  - Sorterad lista är effektivare för de binära mängdoperationerna.
    - Grundmängden måste gå att sortera.

Konstruktion av mängd som bitvektor (1)

- En bitvektor är en vektor med elementvärden av typen `Bit` = {0,1}.
- Ofta tolkas 0=falskt och 1=sant, dvs `Bit` identifieras som datatypen `Boolean`.
- Grundmängden måste ha en diskret linjär ordning av elementen (man kan numrera dem).
- Bit k i bitvektorn motsvarar det k:te elementet i grundmängden.
  - Biten är 1 om elementet ingår i mängden.

Konstruktion av mängd som bitvektor (2)

- Veckoschema måndag-fredag timvis, 08-17 i 64-bitars heltal:

	Mon	Tue	Wed	Thu	Fri
08-09	X			X	
09-10	X			X	
10-11					
11-12					
12-13					
13-14	X			X	
14-15	X			X	
15-16					
16-17					

- Om bitarna räknas från höger till vänster:

```
3210987654321098765432109876543210987654321098765432109876543210
~~~6~~~~~~~~~5~~~~~~~~~4~~~~~~~~~3~~~~~~~~~2~~~~~~~~~1~~~~~~~~~0

- - - - - - - - - -FFFFFFFFFTTTTTTTTTWWWWWWWWWTTTTTTTTTMMMMMMMMM

0000000000000000000000000000011000110000000000000000001100011
```

  - IsBusy(day, hour, word):
    - `BitGet(word, (day-Mon)*9+(hour-8))`
  - Clash(word1, word2):
    - `And(word1, word2)`

Konstruktion av mängd som bitvektor (3)

- Sökoperationer och binära operationer har komplexitet $O(M)$, där M är antalet element i grundmängden.
- Om bitvektorn finns implementerad som ett eller flera *ord* kan man utnyttja maskinoperationer.
  - Processorn gör operationen samtidigt på alla element i vektorn.
  - Detta gör många metoder effektiva.
  - Ex. för en ordlängd på 64 bitar=8 bytes så tar `AND` för 64 bitar samma tid som `AND` för 1 bit.

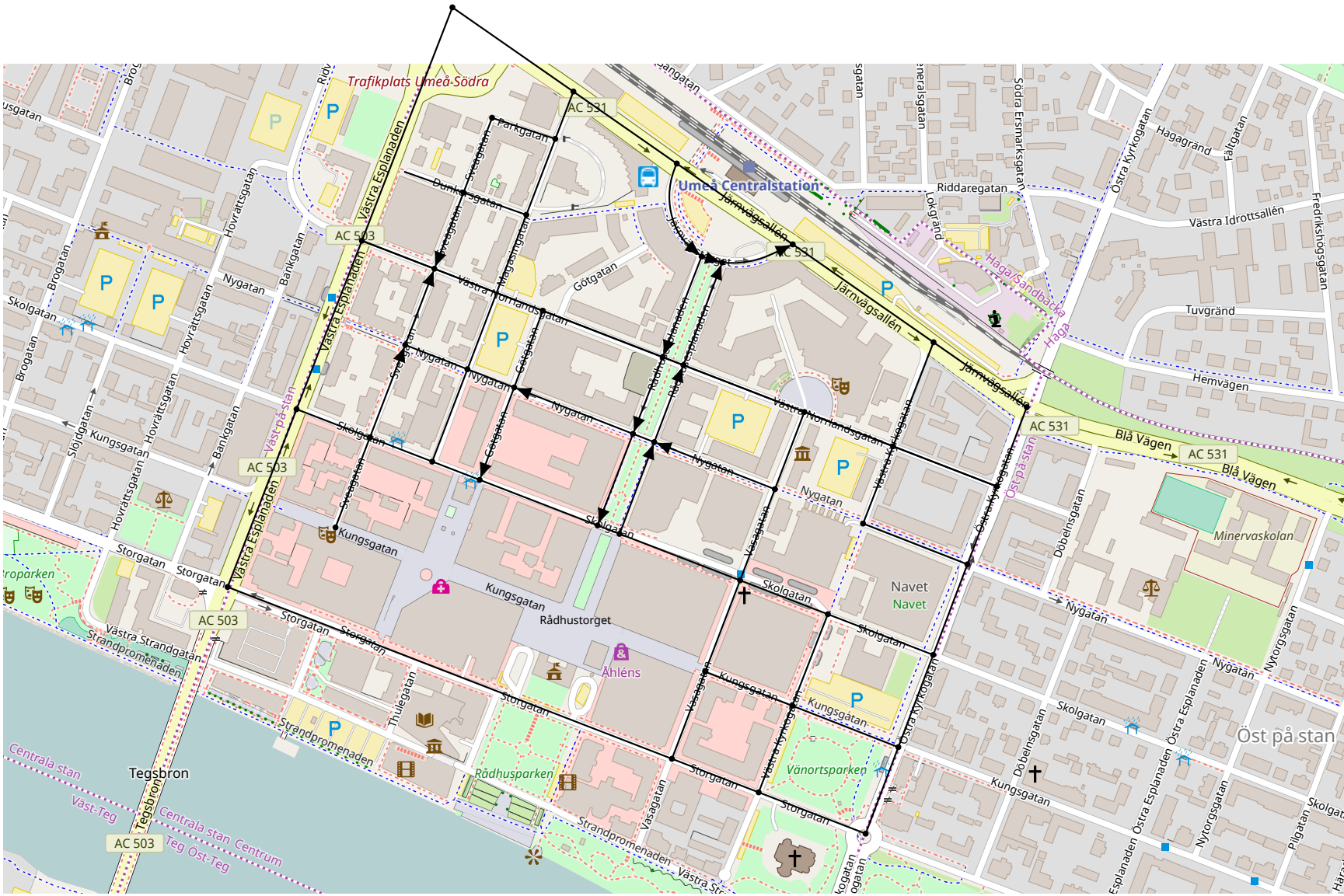
Konstruktion av mängd som bitvektor (4)

- Grundmängden måste vara ändlig och i praktiken liten.
- Reserverat minne proportionellt mot grundmängdens storlek.
  - Ett veckoschema måndag-fredag, uppdelat i timmar 08-17 kräver $M = 45$ bitar (9h per dag, 5 dagar) = 3 bytes.
  - Ett veckoschema 24/7 uppdelat i 5-minutersintervall kräver $M = 2016$ bitar $(7 \cdot 24 \cdot 12)$=252 bytes.
  - En tabell över upptagna IP4-internetadresser kräver $M = 2^{32} = 4294967296$ bitar $\approx$ 0.5Gbyte.
- Få element per mängd utnyttjar minnet ineffektivt.

Graf

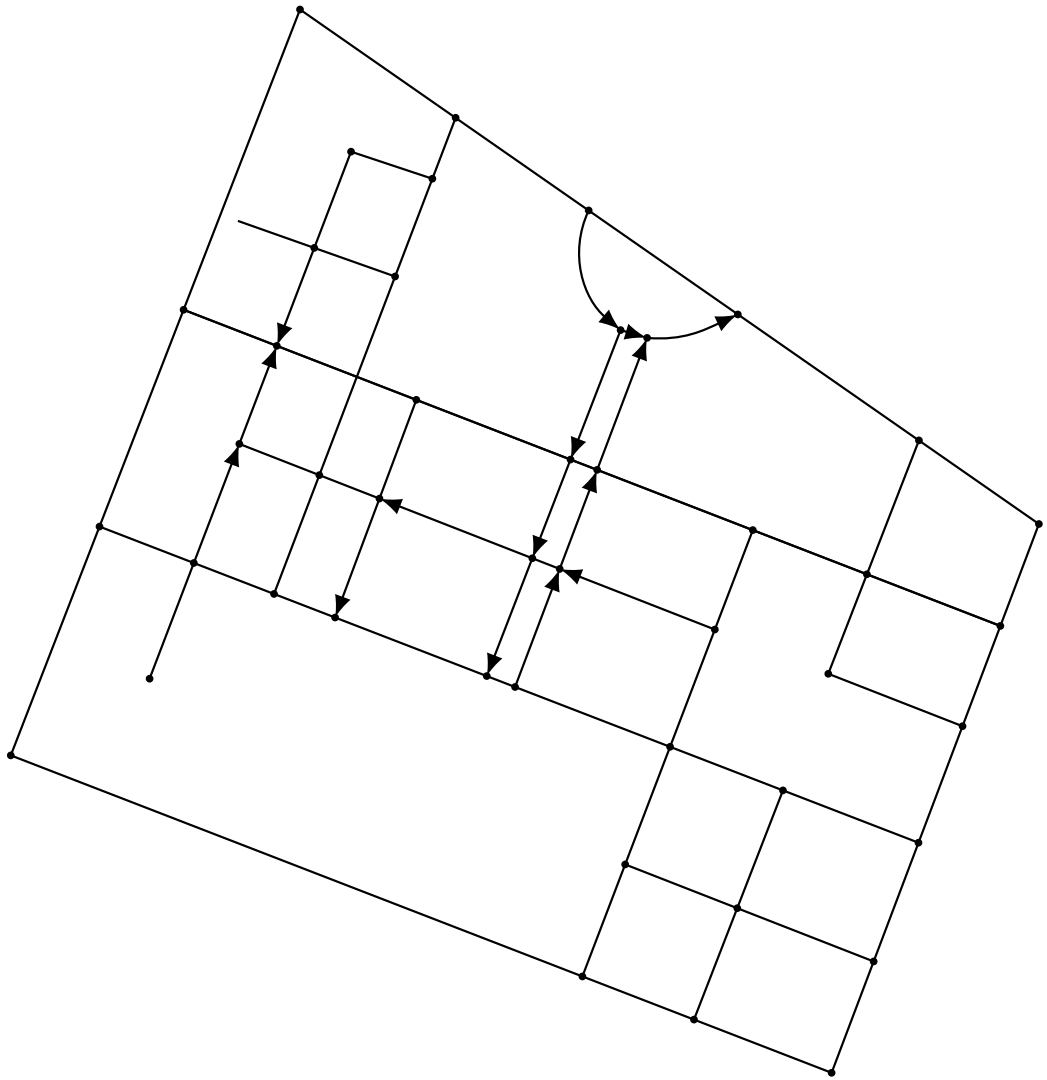
Graf

- Modell: Vägkarta med enkelriktade gator utritade.

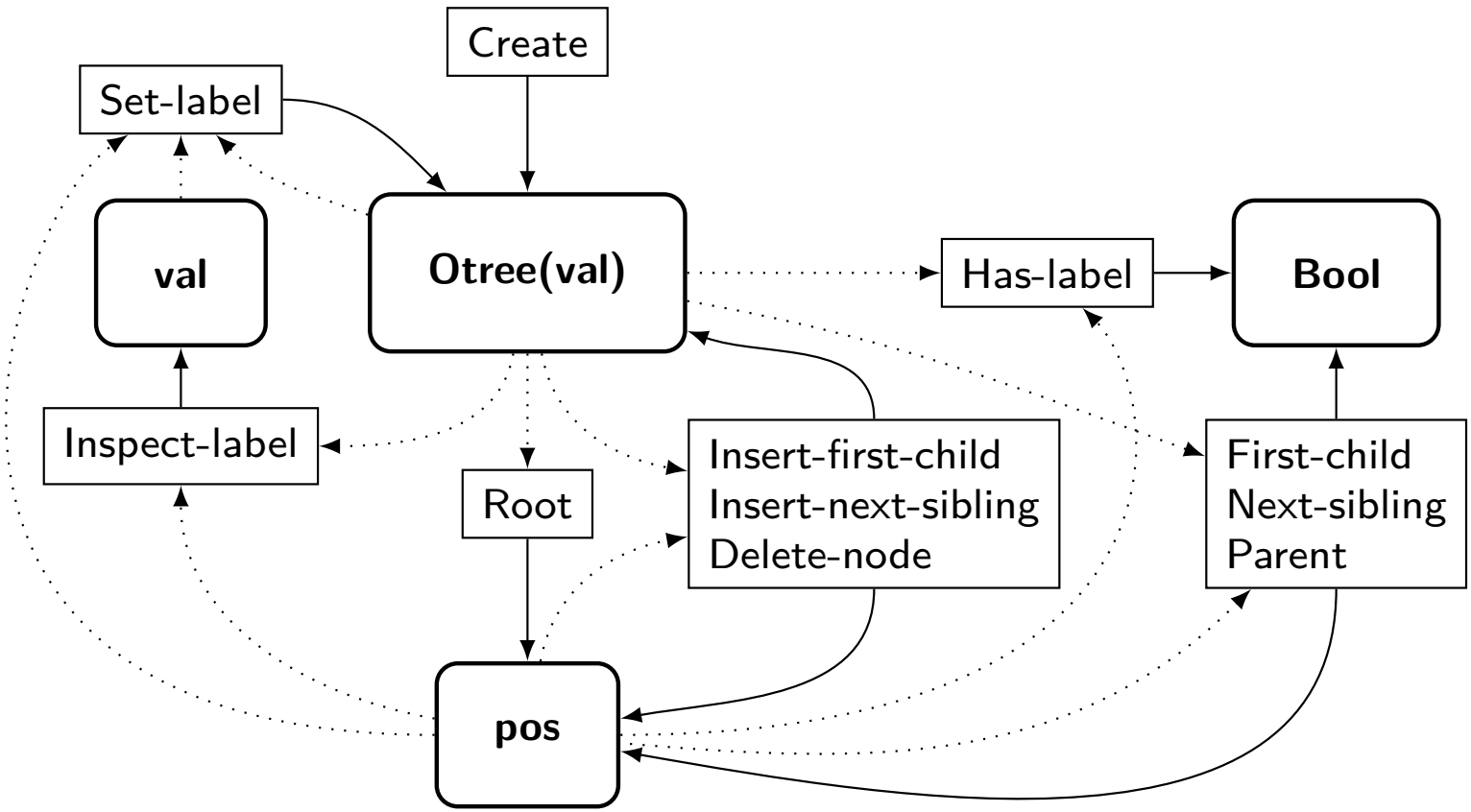


Graf

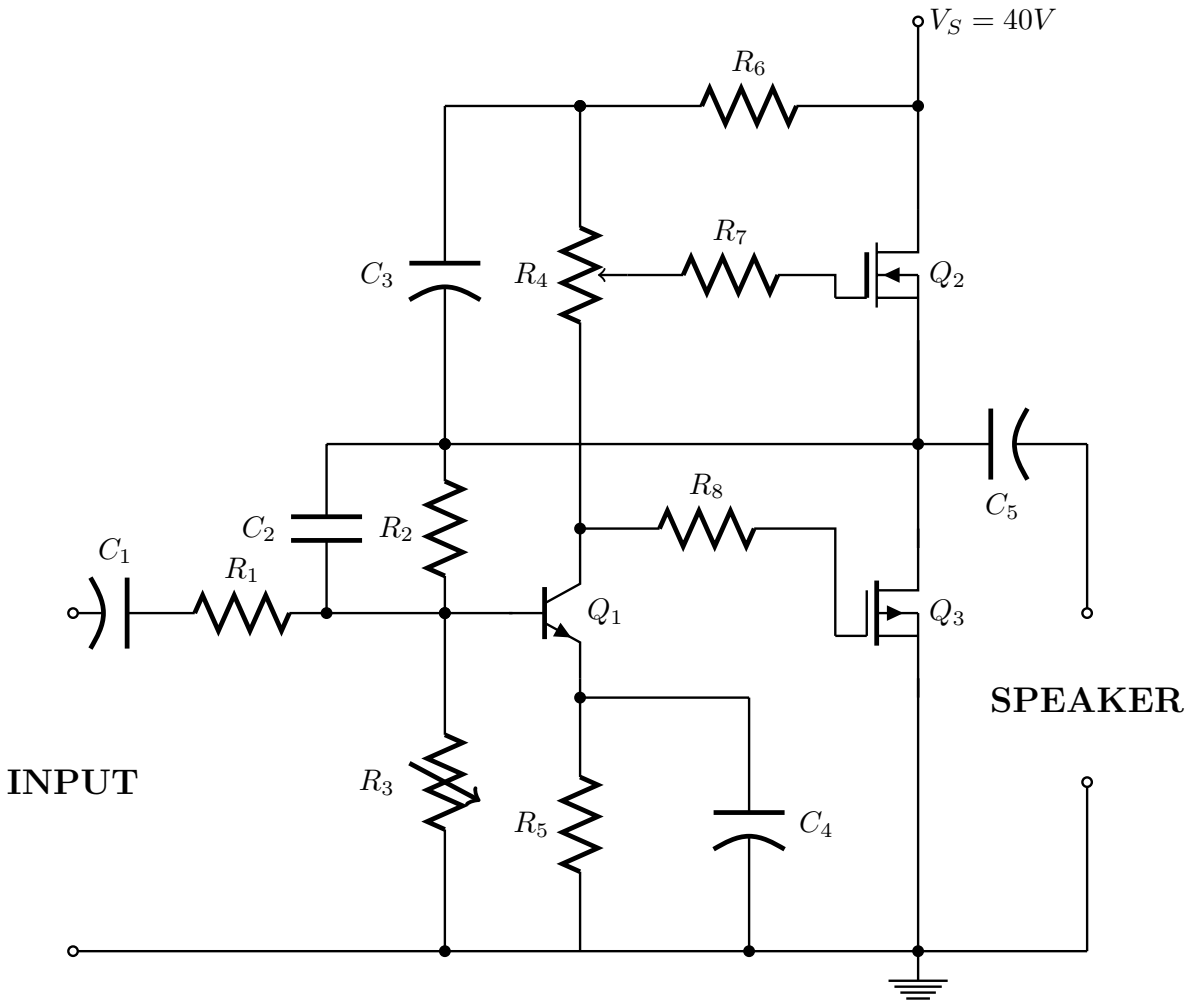
- Modell: Vägkarta med enkelriktade gator utritade.



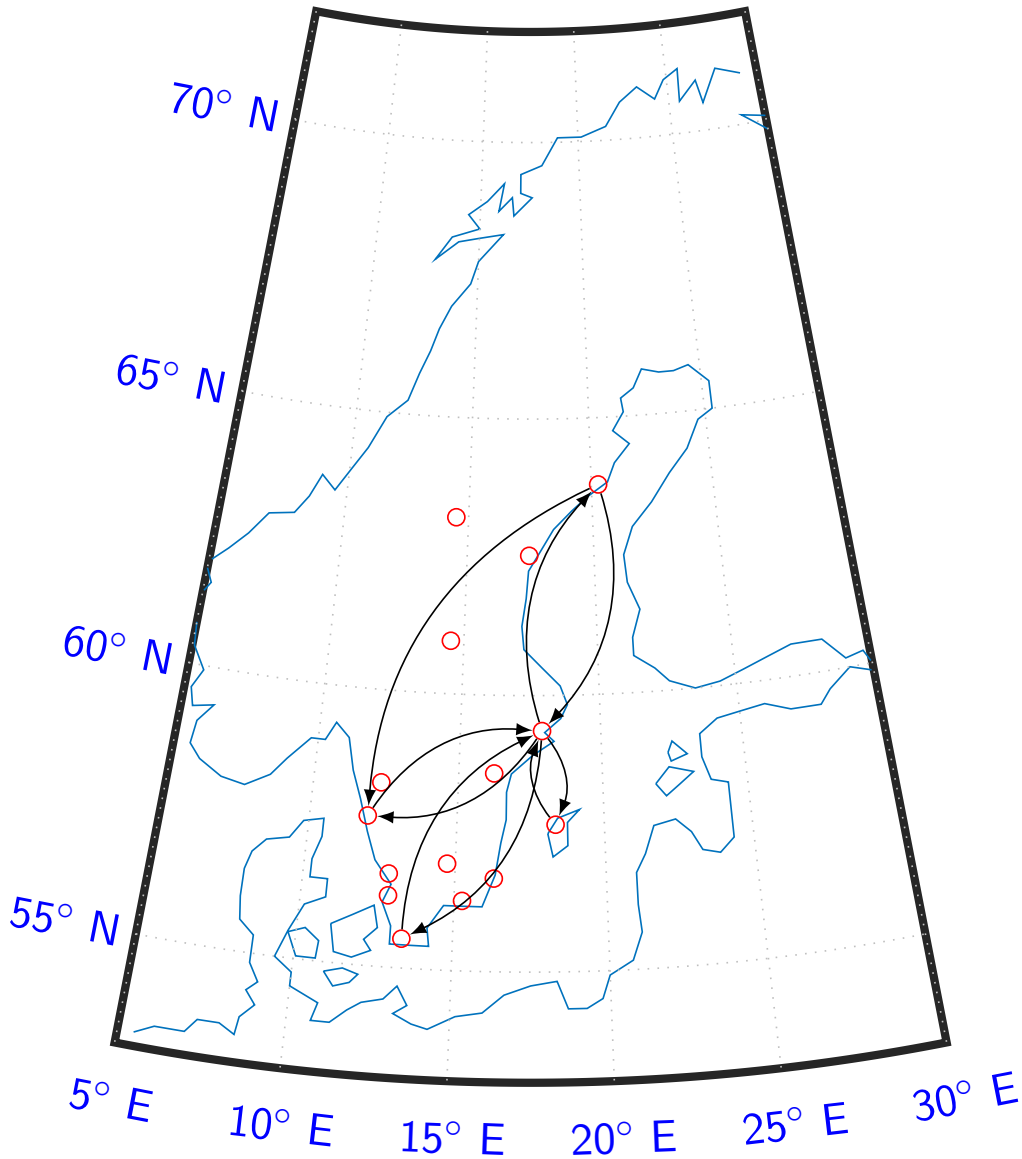
Graf, tillämpningar — signaturdiagrammen



Graf, tillämpningar — elektriska kretsar



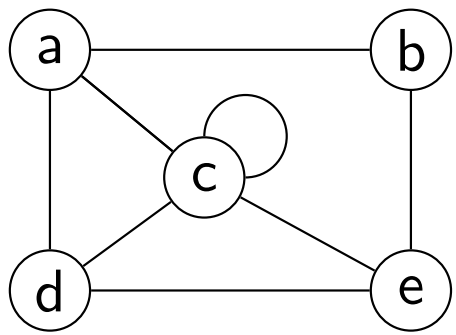
Graf, tillämpningar — nätverk (gator, flygrutter, kommunikation)



Mängorienterad specifikation (vanlig inom matematiken)

- En graf $G = (V, E)$ består av:
  - V en mängd av noder (*vertices*) och
  - E en mängd av bågar (*edges*) som binder samman noderna i V.
    - En båge $e = (u, v)$ är ett par av noder.
- Exempel:

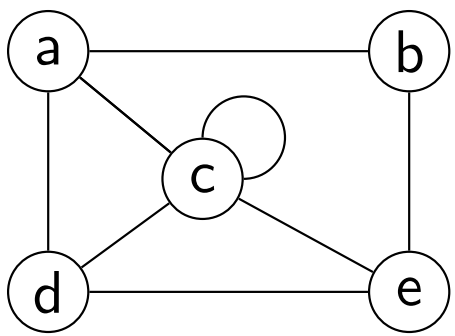
$$V = \{a, b, c, d, e\},$$
$$E = \{(a, b), (a, c), (a, d), (b, e), (c, c), (c, d), (c, e), (d, e)\}$$



Navigeringsorienterad specifikation

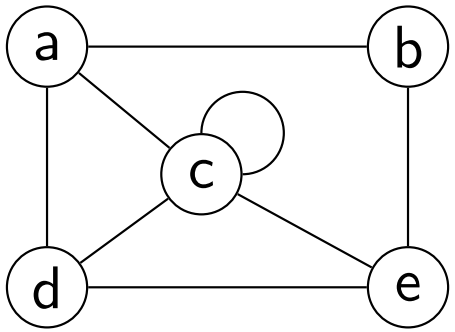
- En graf är en mängd med *noder*.
  - Till varje nod associeras en *grannskapsmängd* av noder som kallas *grannar*.
- Alla noder är av samma typ.
- Alla ordnade par av en godtycklig nod och en av noderna i dess grannskapsmängd utgör en *båge*.
- Navigationsorienterad specifikation effektivare för många algoritmer.
- Exempel:

$$
\begin{aligned}
G = \{\ & (a, \{b, c, d\}) \\
& (b, \{e, a\}) \\
& (c, \{c, a, e, d\}) \\
& (d, \{e, a, c\}) \\
& (e, \{b, c, d\})\ \}
\end{aligned}
$$



Riktade/oriktade grafer

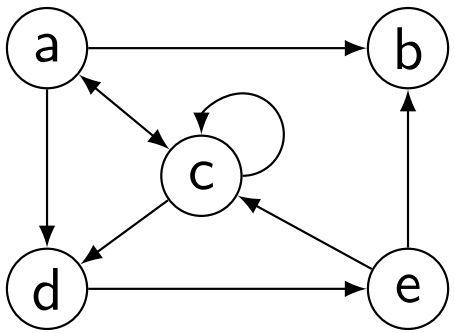
- Oriktade grafer:
  - Bågen är en *mängd* av två noder.
  - Noderna är grannar till varandra.
  - Gradtalet = Antalet bågar till grannar (inklusive sig själv).



- Riktade grafer:
  - Bågen är ett *ordnat par* av noder.
  - Gradtalet indelas i

    Ingradtalet antalet bågar som går *till* noden.

    Utgradtalet antalet bågar som startar i noden och går till en *annan* nod.



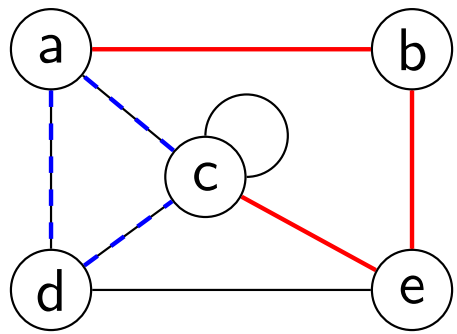
Tänkbar informell specifikation

`abstract datatype Graph`

`Empty()`	—	konstruerar en tom graf utan noder och bågar
`Insert-node(v,g)`	—	sätter in noden v i grafen g
`Insert-edge(e,g)`	—	sätter in en båge e i grafen g. Det förutsätts att noderna finns i grafen.
`Isempty(g)`	—	testar om grafen g är tom, dvs. utan noder
`Has-no-edges(g)`	—	testar om grafen g saknar bågar
`Choose-node(g)`	—	väljer ut en godtycklig nod ur grafen g
`Neighbours(v,g)`	—	mängden av alla granner till v i grafen g
`Delete-node(v,g)`	—	tar bort noden v ur grafen g, förutsatt att v inte ingår i nån båge
`Delete-edge(e,g)`	—	tar bort bågen e ur grafen g

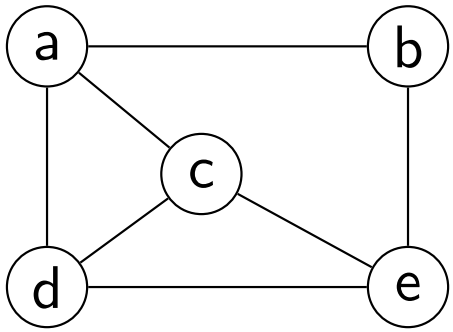
Terminologi

- Väg/stig (*path*): En sekvens av noder $v_1, v_2, ..., v_n$ så att v_i och v_{i+1} är grannar.
  - Sekvenserna $a - b - e - c$ och $a - c - d - a$ är vägar.
- Enkel väg (*simple path*): Inga noder förekommer två gånger i vägen.
  - Vägen $a - b - e - c$ är en enkel väg
- Cykel (*cycle*): En enkel väg där den sista noden i sekvensen är densamma som den första.
  - Vägen $a - c - d - a$ är en cykel.

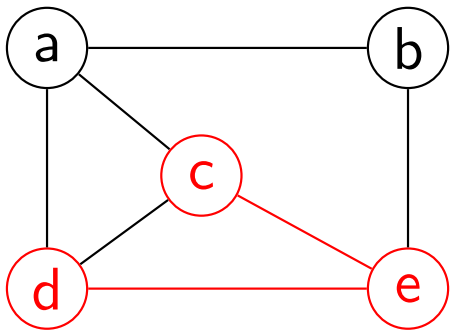


Sammanhängande/icke sammanhängande grafer

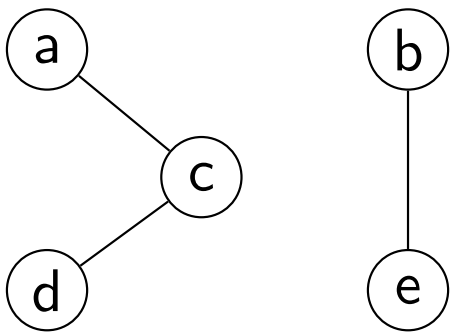
- Sammanhängande (*connected*) graf:
  - Varje nod har en väg till varje annan nod.



- Delgraf (subgraf):
  - En delmängd av noderna och kanterna som formar en graf.

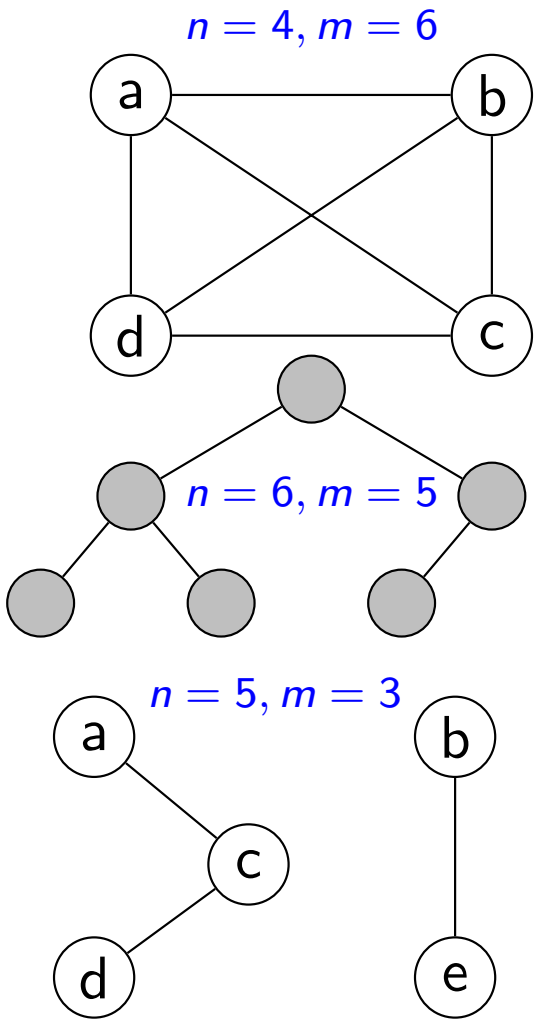


- Icke sammanhängande med sammanhängande komponenter.
  - Grafen till höger har två sammanhängande komponenter.



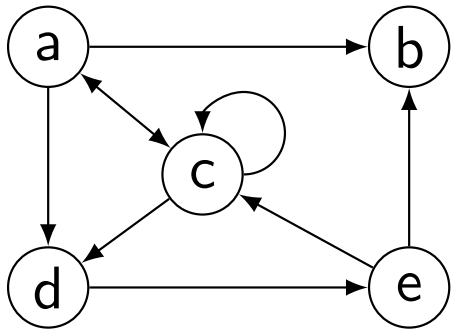
Nåbarhet (*Connectivity*)

- Låt n = antalet noder och m = antalet bågar.
- En komplett graf (*complete graph*) får man när alla noder är grannar till alla andra.
- I en komplett oriktad graf är $m = n(n-1)/2$.
- För ett träd gäller $m = n - 1$.
- Om $m < n - 1$ så är grafen inte sammanhängande.

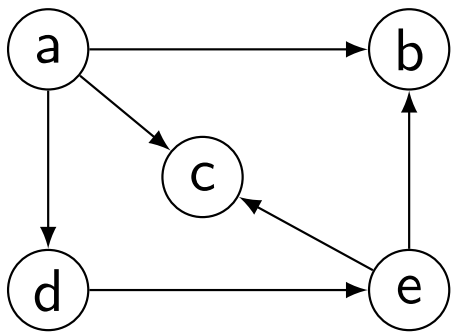


Digraph och DAGs

- DiGraph = *Directed graph* (riktad graf).
  - Grannrelationen ej symmetrisk, dvs. *b* kan vara granne till *a* utan att *a* är granne till *b*.



- DAG = *Directed Acyclic Graph*:
  - En riktad graf utan cykler.

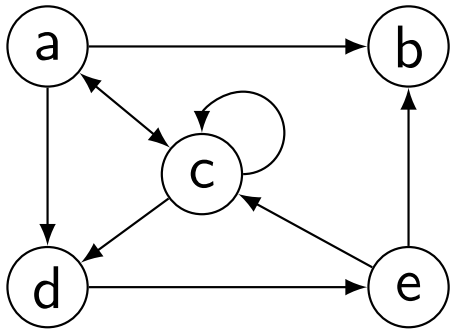


Mer grafer

- Viktad graf:
  - En graf där bågarna har vikter.
- Multigraf:
  - Tillåtet med flera bågar mellan två noder.
  - Bågarna har olika egenskaper som måste lagras.
- Ordnad graf:
  - Grannarna är ordnade.

Konstruktion av grafer, förbindelsematris (*adjacency matrix*)

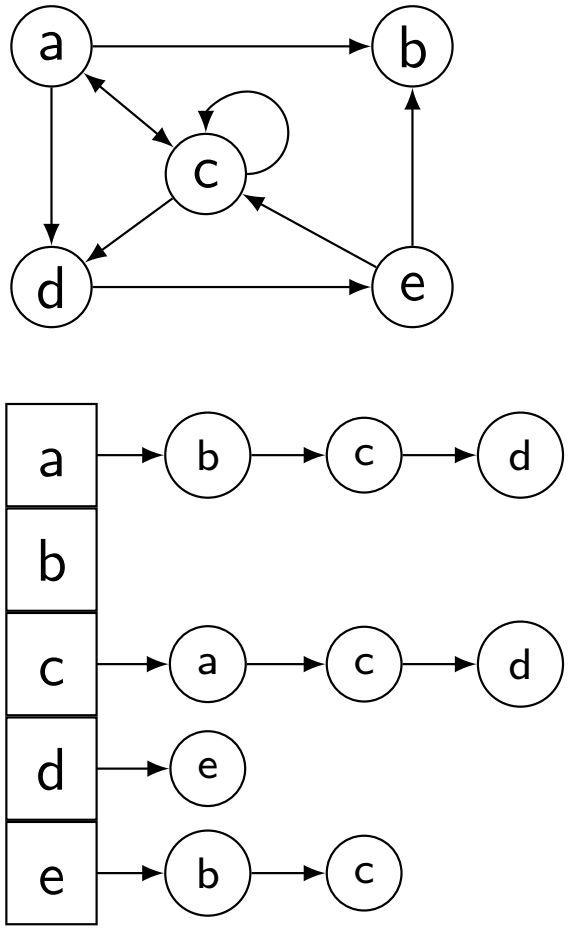
- Bågarna representeras av ettor i en matris.
  - Rad i visar vilka bågar man kan nå *från* nod i.
  - Kolumn j visar från vilka noder det kommer bågar *till* nod j.
- Enkel att implementera.
- Passar också när man har vikter på bågar.
- Matrisen kan bli stor.
  - Minne: $O(n^2)$.
- (Google använder en matris med n=antalet sidor på internet för att rangordna sina sökresultat.)



	a	b	c	d	e
a	0	1	1	1	0
b	0	0	0	0	0
c	1	0	1	1	0
d	0	0	0	0	1
e	0	1	1	0	0

Konstruktion av grafer, fält av lista

- Fältelementen innehåller en nod och en lista — grannskapslistan.
- Antalet noder fixt (fält), antalet grannar per nod variabelt (lista).
- Inte lika utrymmeskrävande som en gles matris. Utrymmet $O(m + n)$.



Obligatorisk uppgift 5

- Börja i tid med uppgiften!
- Redan nu kan ni t.ex. börja fundera på inläsning från fil och hur ni vill konstruera grafen.
- Algoritmerna för traversering kommer att gås igenom på nästa föreläsning.