Grafer

I. GRAF

definisjon / terminologi

noen eksempler på graf-problemer

II. NOEN ENKLE GRAFALGORITMER

III. GRAF TRAVERSERING

DFS og BFS

IV. GRAF ADT OG IMPLEMENTASJON

Kant-Liste og Nabo-Liste

Nabo-Matrise

V. RETTEDE GRAFER

topologisk sorterting, transitiv tillukking

VI. VEKTEDE GRAFER

kortest sti, minste utspennede tre

Kap. 12 (kursorisk 12.4.4, 12.7.2)

Grafer

I. GRAF

definisjon / terminologi

noen eksempler på graf-problemer

II. NOEN ENKLE GRAFALGORITMER

III. GRAF TRAVERSERING

DFS og BFS

IV. GRAF ADT OG IMPLEMENTASJON

Kant-Liste og Nabo-Liste

Nabo-Matrise

V. RETTEDE GRAFER

topologisk sorterting, transitiv tillukking

VI. VEKTEDE GRAFER

kortest sti, minste utspennede tre

Kap. 12 (kursorisk 12.4.4, 12.7.2)

I. En graf G

er gitt ved to mengder V og E; G=(V,E)

V er noder (boka: vertices)

E er kanter (boka: edges)

en kant e ε E er et (uordnet)

par {u,v} av noder u ε V og v ε V

I. En graf G

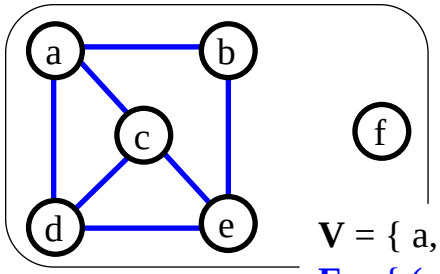
er gitt ved to mengder V og E; G=(V,E)

V er noder (boka: vertices)

E er kanter (boka: edges)

en kant e ε E er et (uordnet)

par {u,v} av noder u ε V og v ε V

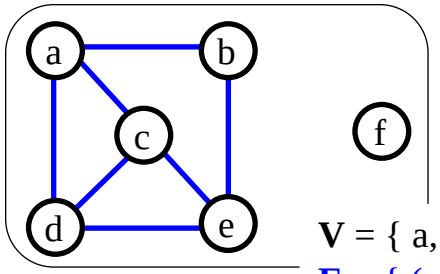


V = { a, b, c, d, e, f }

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

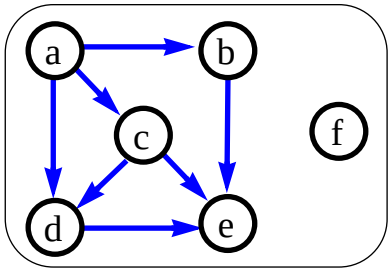
I. En graf G

er gitt ved to mengder V og E; G=(V,E)
V er noder (boka: vertices)
E er kanter (boka: edges)
en kant e ε E er et (uordnet)
par {u,v} av noder u ε V og v ε V



V = { a, b, c, d, e, f }
E = { (a,b), **(a,c)**, (a,d), (b,e),
(c,d), (c,e), (d,e) }
= { (b,a), **(c,a)**, (d,a), (e,b),
(d,c), (e,c), (e,d) }

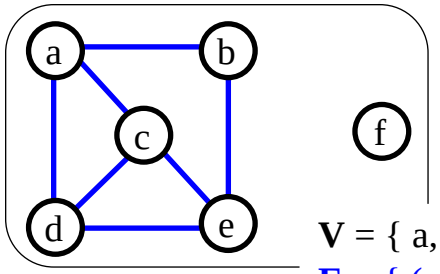
En **rettet graf** (diGraf)
V av noder
E av kanter
der en kant e ε E er et ordnet
par (u,v), u ε V, v ε V, dvs u → v



E = { (a,b), **(a,c)**, (a,d), (b,e),
(c,d), (c,e), (d,e) }

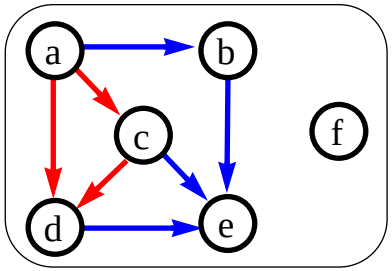
I. En graf G

er gitt ved to mengder V og E; G=(V,E)
V er noder (boka: vertices)
E er kanter (boka: edges)
en kant e ε E er et (uordnet)
par {u,v} av noder u ε V og v ε V

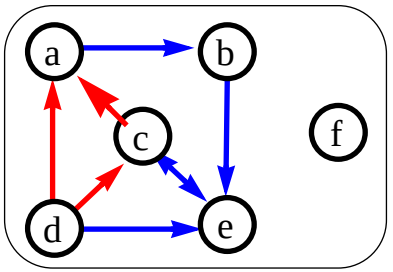


V = { a, b, c, d, e, f }
E = { (a,b), (a,c), (a,d), (b,e),
(c,d), (c,e), (d,e) }
= { (b,a), (c,a), (d,a), (e,b),
(d,c), (e,c), (e,d) }

En **rettet graf** (diGraf)
V av noder
E av kanter
der en kant e ε E er et ordnet
par (u,v), u ε V, v ε V, dvs u → v



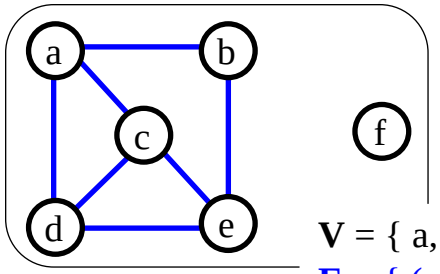
E = { (a,b), **(a,c)**, **(a,d)**, (b,e),
(c,d), (c,e), (d,e) }



E = { (a,b), **(c,a)**, **(d,a)**, (b,e),
(d,c), (c,e), (d,e) }

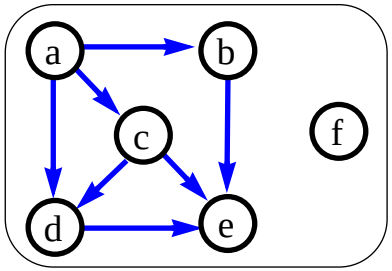
I. En graf G

er gitt ved to mengder V og E; G=(V,E)
V er noder (boka: vertices)
E er kanter (boka: edges)
en kant e ε E er et (uordnet)
par {u,v} av noder u ε V og v ε V

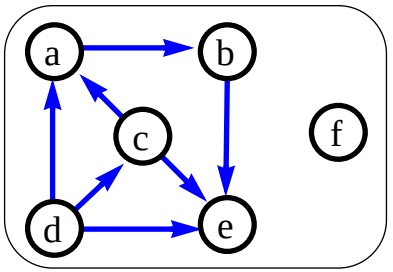


V = { a, b, c, d, e, f }
E = { (a,b), (a,c), (a,d), (b,e),
(c,d), (c,e), (d,e) }
= { (b,a), (c,a), (d,a), (e,b),
(d,c), (e,c), (e,d) }

En **rettet graf** (diGraf)
V av noder
E av kanter
der en kant e ε E er et ordnet
par (u,v), u ε V, v ε V, dvs $u \rightarrow v$



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

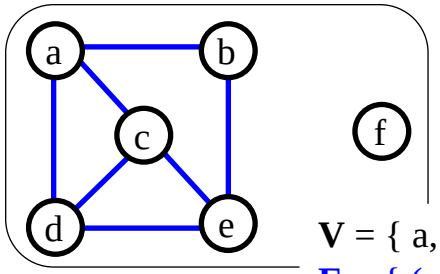


E = { (a,b), **(c,a)**, **(d,s)**, (b,e),
(d,c), (c,e), (d,e) }

En (rettet) **graf** er gitt ved
en mengde V av noder og
en **binær relasjon** E $\subseteq V \times V$

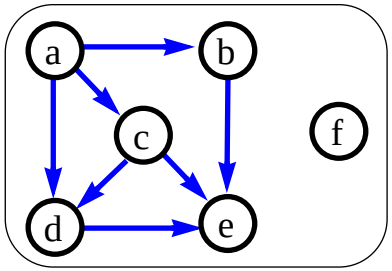
I. En graf G

er gitt ved to mengder V og E; G=(V,E)
V er noder (boka: vertices)
E er kanter (boka: edges)
en kant e ε E er et (uordnet)
par {u,v} av noder u ε V og v ε V

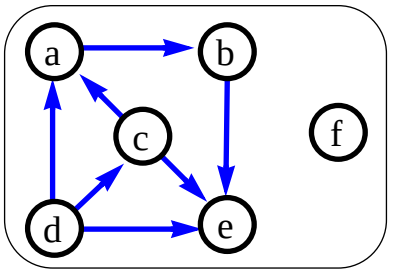


V = { a, b, c, d, e, f }
E = { (a,b), (a,c), (a,d), (b,e),
(c,d), (c,e), (d,e) }
= { (b,a), (c,a), (d,a), (e,b),
(d,c), (e,c), (e,d) }

En **rettet graf** (diGraf)
V av noder
E av kanter
der en kant e ε E er et ordnet
par (u,v), u ε V, v ε V, dvs u → v



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



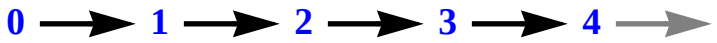
E = { (a,b), **(c,a)**, **(d,s)**, (b,e),
(d,c), (c,e), (d,e) }

En **ikke-rettet graf** kan sees på som en (rettet) graf der relasjonen E er **symmetrisk** u → v ε E *hviss* v → u ε E.

En (rettet) **graf** er gitt ved
en mengde V av noder og
en **binær relasjon** E ⊆ V × V

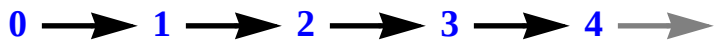
Anvendelser ...

I. BINÆRE RELASJONER

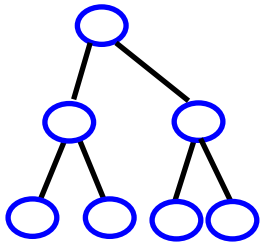


Anvendelser ...

I. BINÆRE RELASJONER



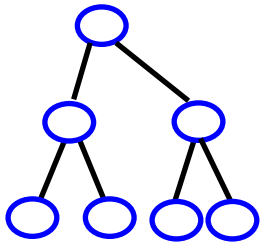
II. TRÆR



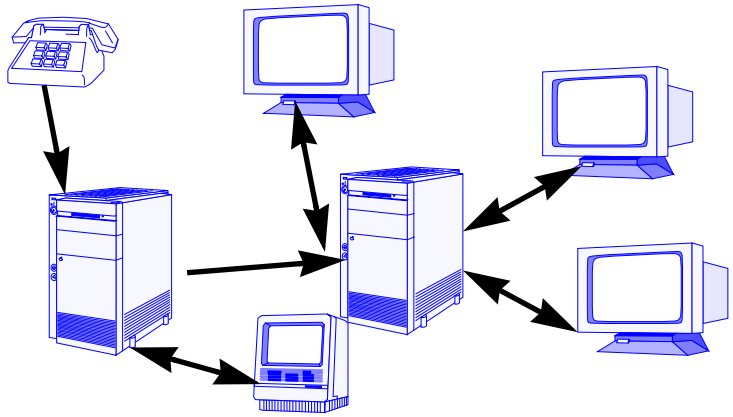
Anvendelser ...

I. BINÆRE RELASJONER

II. TRÆR

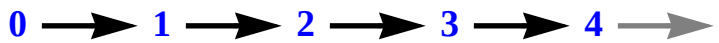


III. NETTVERK

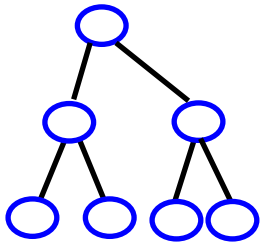


Anvendelser ...

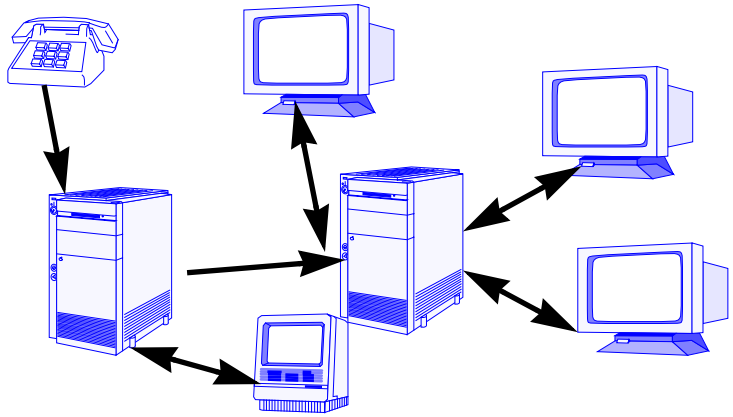
I. BINÆRE RELASJONER



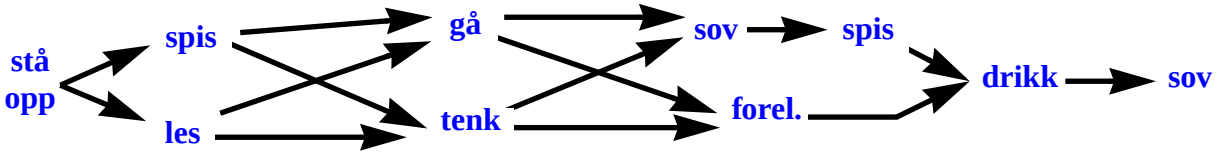
II. TRÆR



III. NETTVERK

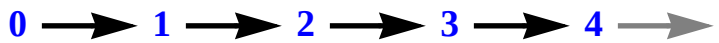


IV. PLANLEGGING

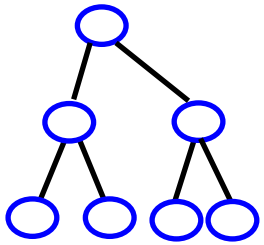


Anvendelser ...

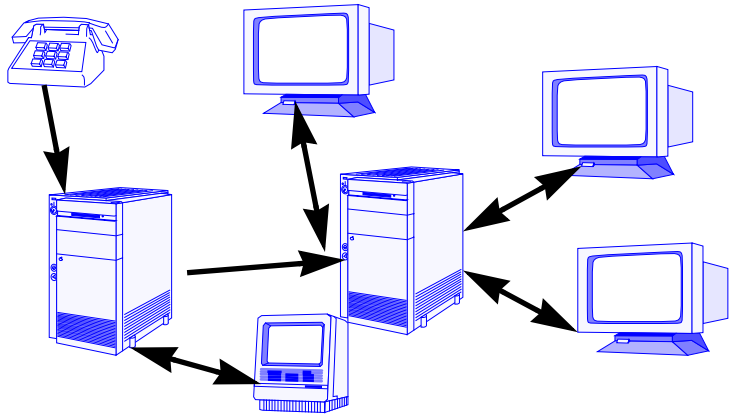
I. Binære relasjoner



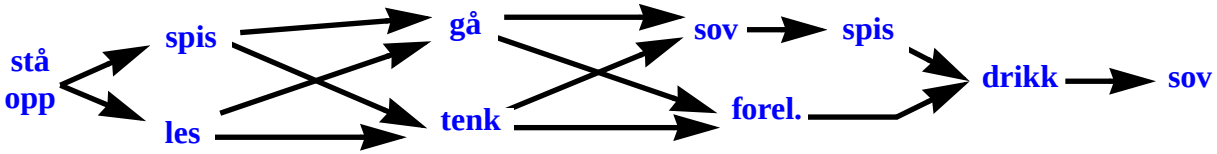
II. Trær



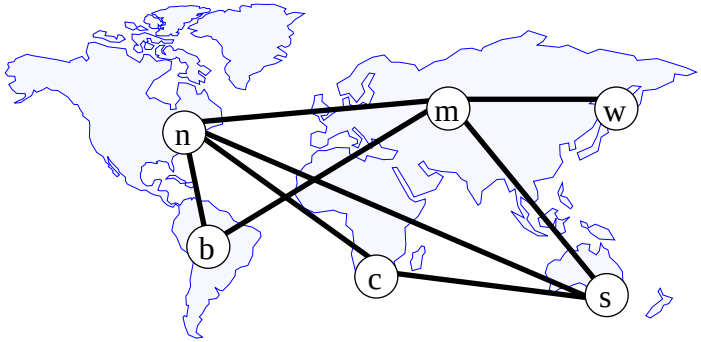
III. Nettverk



IV. Planlegging



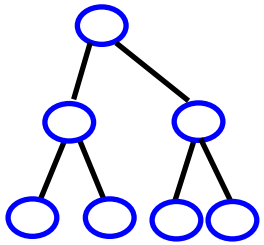
V. Forbindelser



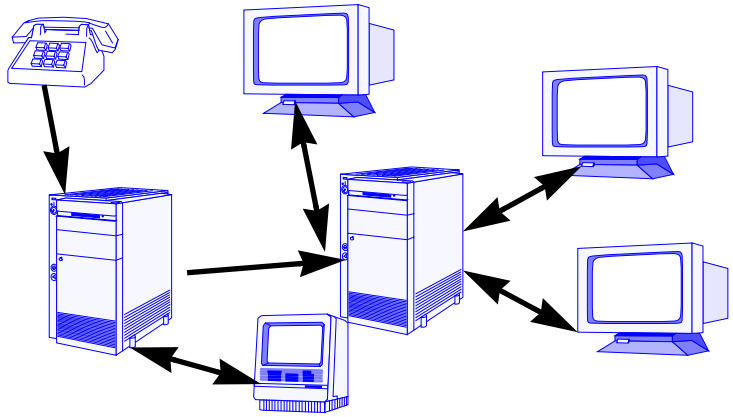
Anvendelser ...

I. Binære relasjoner

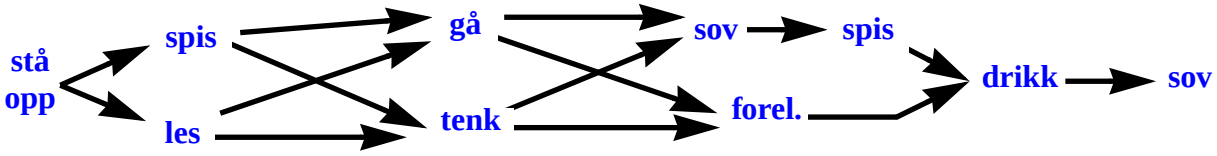
II. Trær



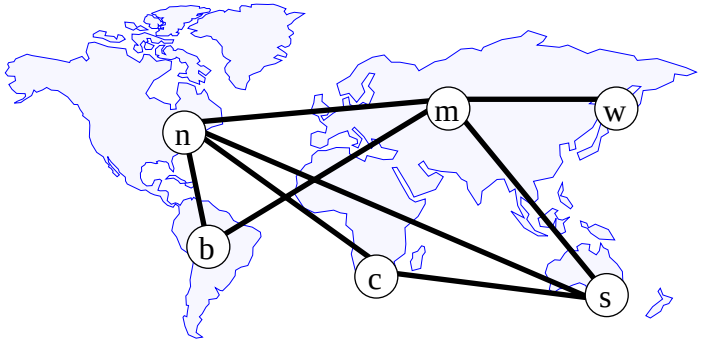
III. Nettverk

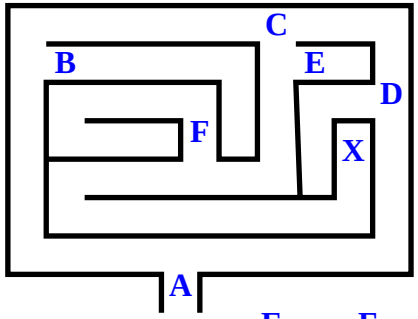


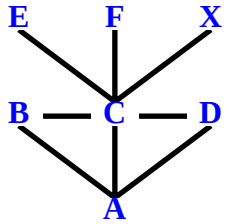
IV. Planlegging



V. Forbindelser



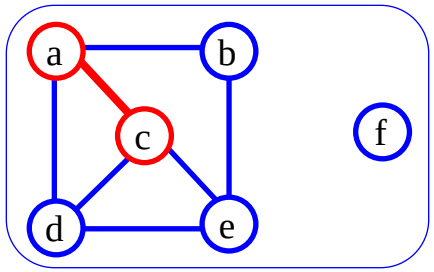




. . .

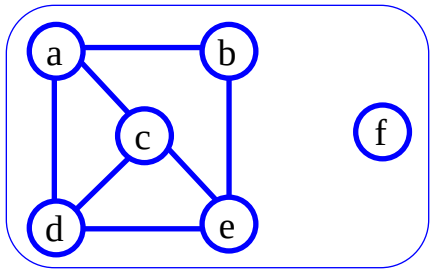
Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant a-c, c-e



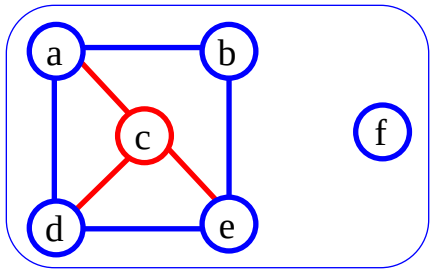
Graf-terminologi

nabonoder :	2 noder som er forbundet med en kant	a-c, c-e
gradtall til en node :	antall nabonoder (kanter):	**deg(c) = 3**, deg(f)=0



Graf-terminologi

nabonoder :	2 noder som er forbundet med en kant	a-c, c-e
gradtall til en node :	antall nabonoder (kanter):	**deg(c) = 3**, deg(f)=0



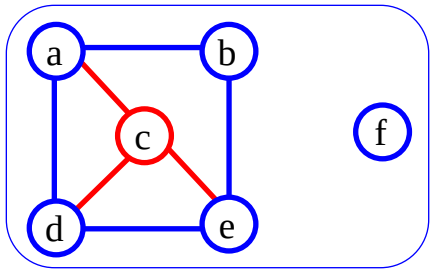
Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c) = 3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

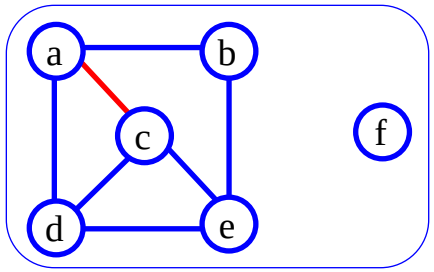
gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens n_1, n_2 ... n_k av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

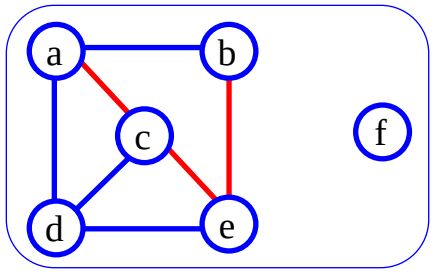
gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens n_1, n_2 ... n_k av noder slik at $(n_i, n_{i+1}) \in E$

acaca, **beca**, abedc, edce



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

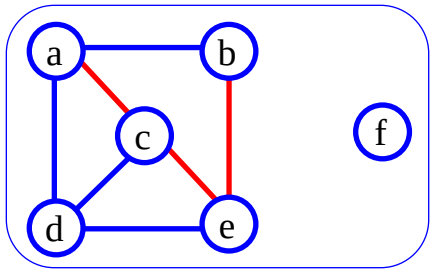
$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, **beca**, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

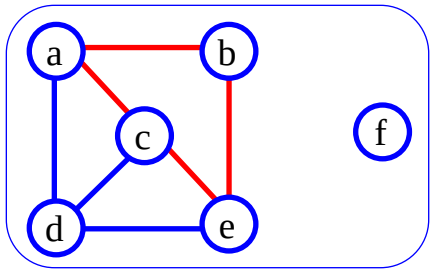
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, beca, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

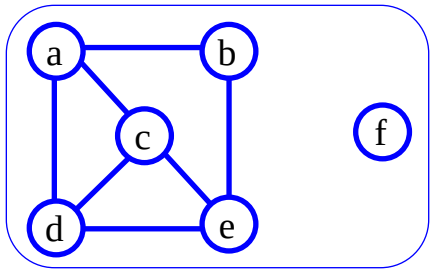
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

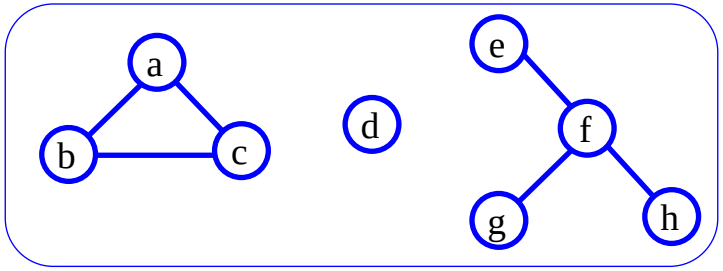
acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

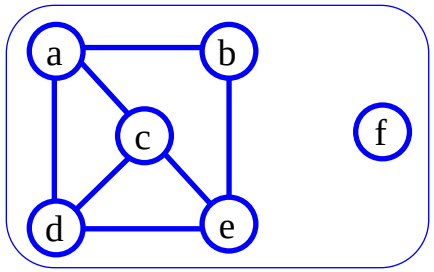
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

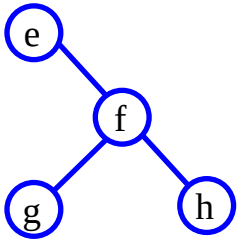
acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

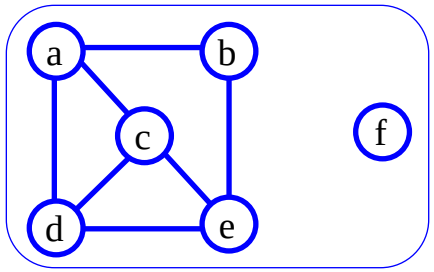
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

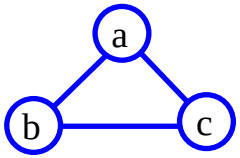
acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

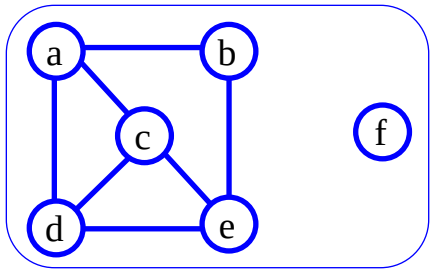
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

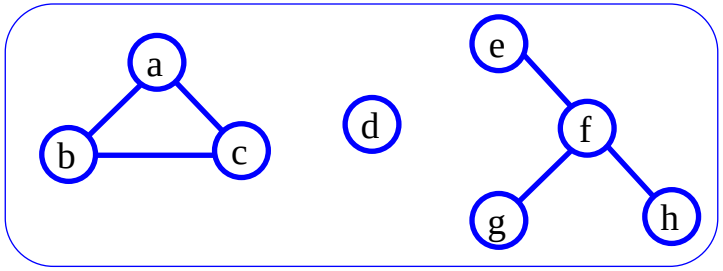
enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

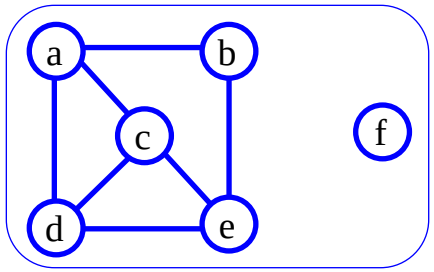
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

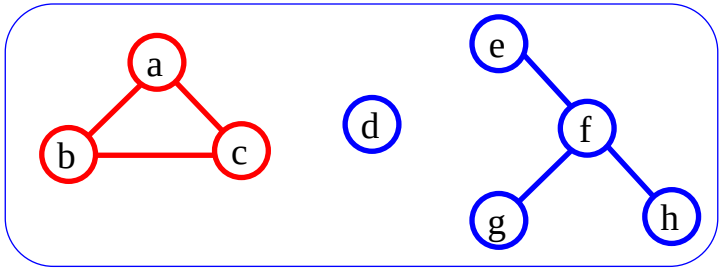
sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

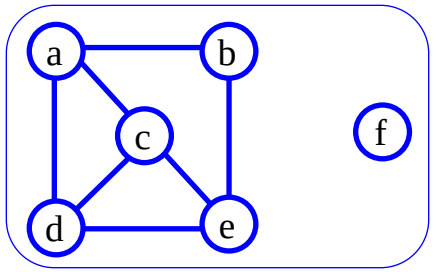
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

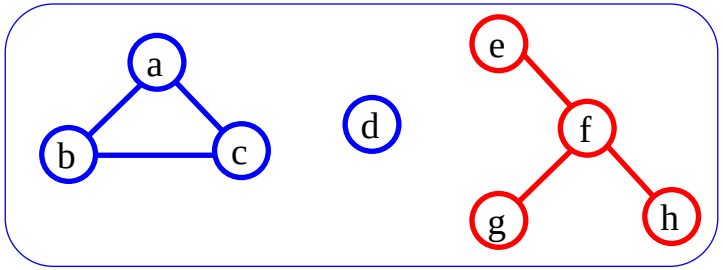
sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

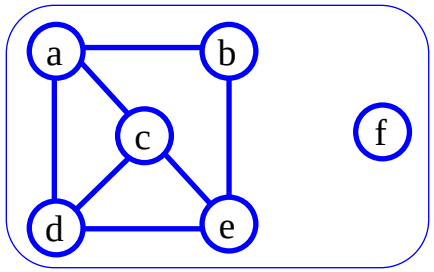
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

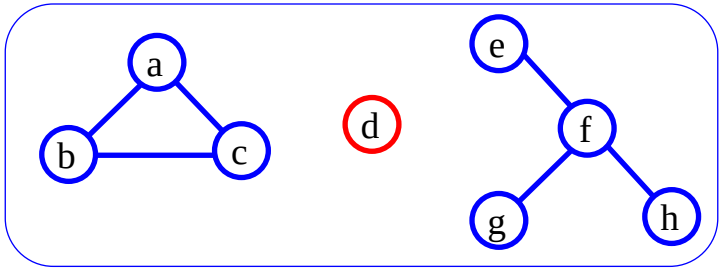
sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf



Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

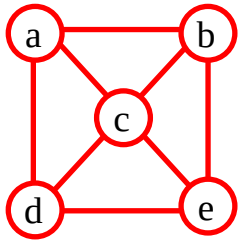
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

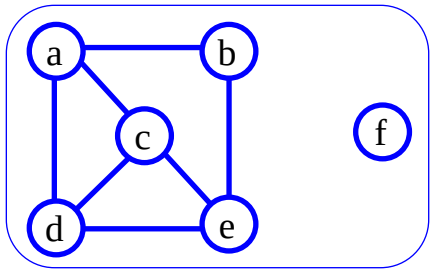
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

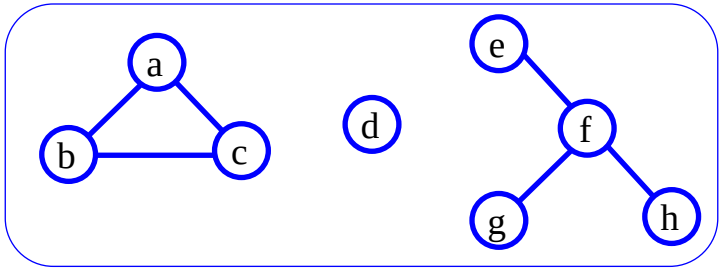
sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf



komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

(ikke-rotet) **tre** : sammenhengende graf uten sykler

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

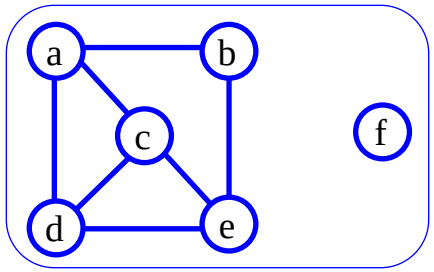
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

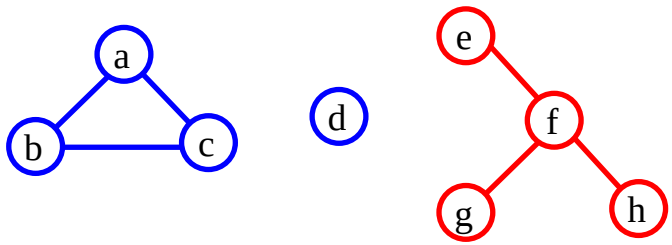
sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf



komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

(ikke-rotet) **tre** : sammenhengende graf uten sykler

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

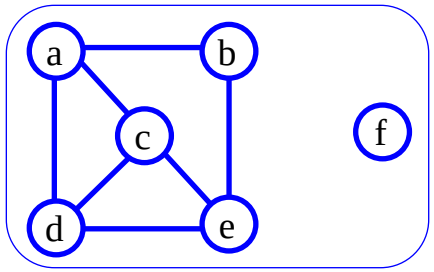
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

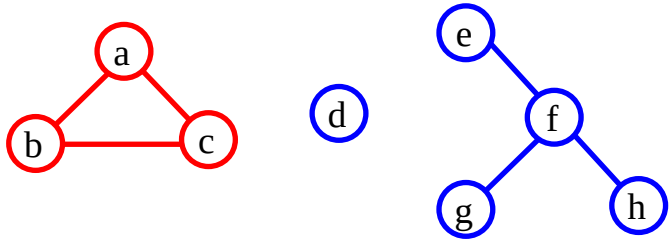
sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf



komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

(ikke-rotet) **tre** : sammenhengende graf uten sykler

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

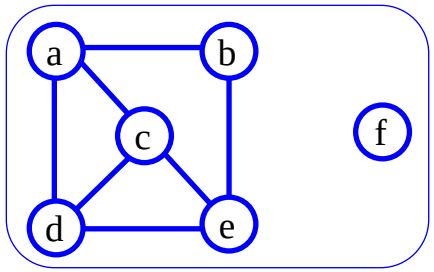
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

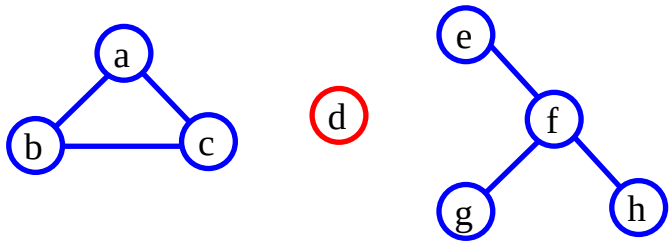
sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf



komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

(ikke-rotet) **tre** : sammenhengende graf uten sykler

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

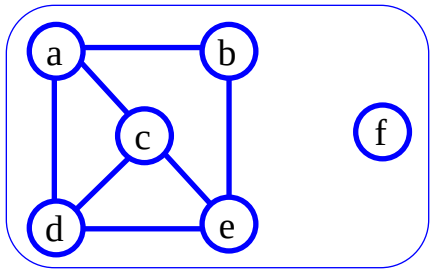
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$

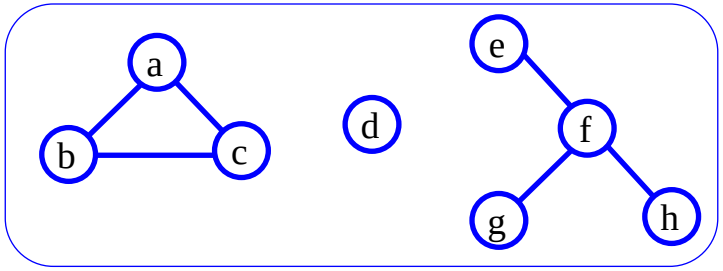


sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E



(ikke-rotet) **tre** : sammenhengende graf uten sykler

utspennende tre for en graf G : en delgraf av G som er et tre og inneholder alle G's noder

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

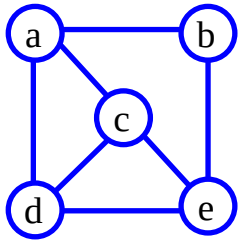
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \in E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

(ikke-rotet) **tre** : sammenhengende graf uten sykler

utspennende tre for en graf G : en delgraf av G som er et tre og inneholder alle G's noder

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

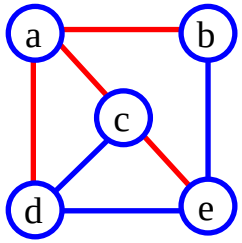
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \in E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

(ikke-rotet) **tre** : sammenhengende graf uten sykler

utspennende tre for en graf G : en delgraf av G som er et tre og inneholder alle G's noder

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

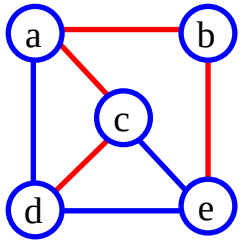
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \in E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$



sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

(ikke-rotet) **tre** : sammenhengende graf uten sykler

utspennende tre for en graf G : en delgraf av G som er et tre og inneholder alle G's noder

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

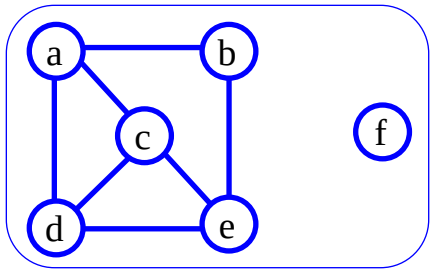
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$

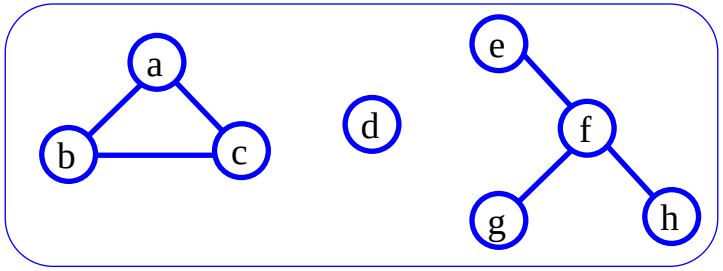


sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E



(ikke-rotet) **tre** : sammenhengende graf uten sykler

utspennende tre for en graf G : en delgraf av G som er et tre og inneholder alle G's noder

skog : samling av trær

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

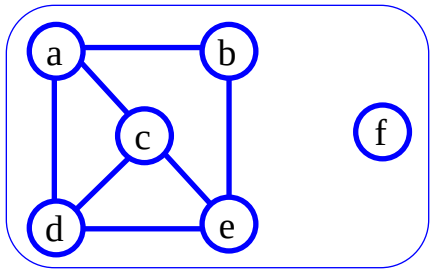
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \varepsilon E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$

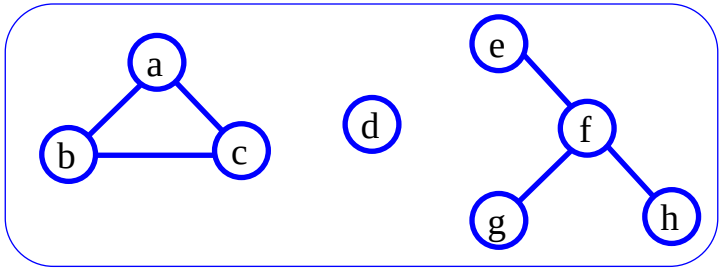


sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

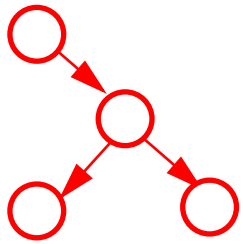
sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E



(ikke-rotet) **tre** : sammenhengende graf uten sykler

utspennende tre for en graf G : en delgraf av G som er et tre og inneholder alle G's noder



skog : samling av trær

DAG :rettet graf uten sykler (directed acyclic graph)

Graf-terminologi

nabonoder : 2 noder som er forbundet med en kant — a-c, c-e

gradtall til en node : antall nabonoder (kanter): — deg(c)=3, deg(f)=0

$$\sum_{v \in V} deg(v) = 2(\text{ \# kanter })$$

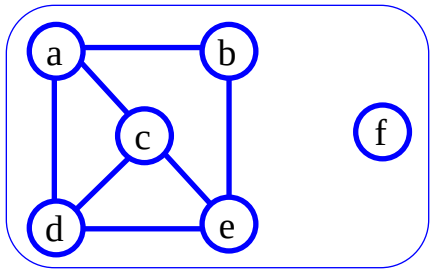
inngrad/utgrad : antall innkommende/utgående kanter (rettede grafer)

sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \in E$

acaca, bec, abedc, edce

enkel sti : en sti hvor ingen node forekommer 2 ganger

sykel : en sti hvor $n_1 = n_k$

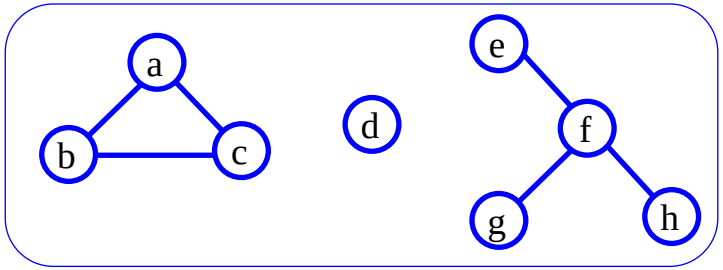


sammenhengende graf : graf hvor det finnes en sti mellom alle par av noder

delgraf : en delmengde av V og E som er en graf

sammenhengende komponent : en maksimal sammenhengende delgraf

komplett graf : om det for hvert par av noder u, v ε V, finnes en kant (u,v) ε E

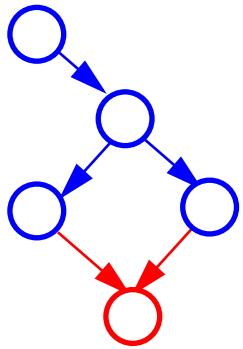


(ikke-rotet) **tre** : sammenhengende graf uten sykler

utspennende tre for en graf G : en delgraf av G som er et tre og inneholder alle G's noder

skog : samling av trær

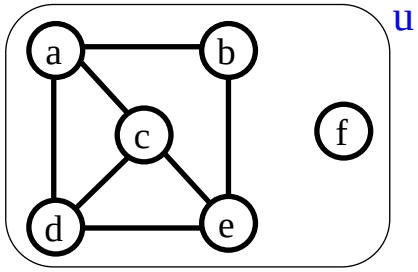
DAG :rettet graf uten sykler (directed acyclic graph)



Graf

URETTET:

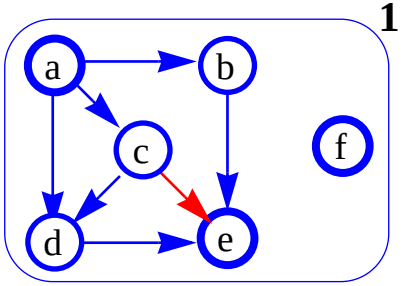
en ikke-rettet kant (u,v)
= to rettede kanter
u → v og v → u



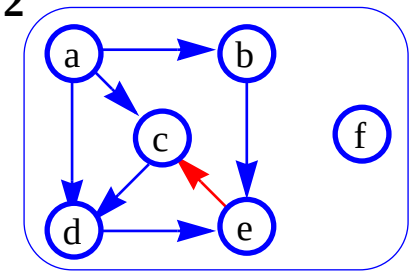
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par u → v.

1



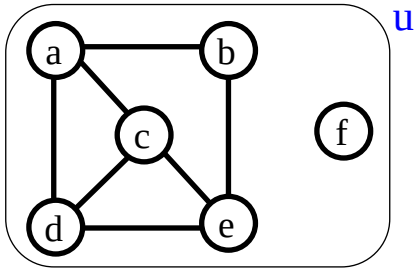
2



Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
$u \rightarrow v$ og $v \rightarrow u$



sti : en sekvens $n_1, n_2 \ldots n_k$ av noder
slik at $(n_i, n_{i+1}) \in E$

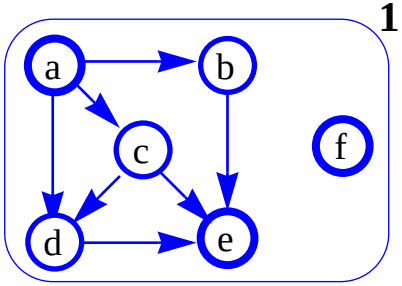
syklus: enkel sti (hver node 1 gang) men $n_1 = n_k$

sammenhengende
graf : det finnes en sti mellom alle par av noder

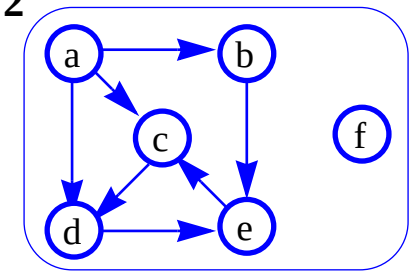
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par $u \rightarrow v$.

1



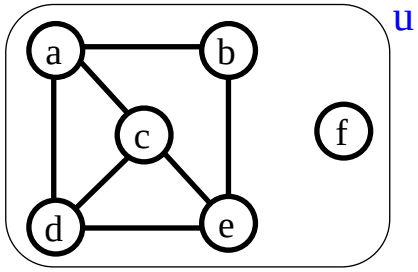
2



Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
$u \rightarrow v$ og $v \rightarrow u$



sti : en sekvens $n_1, n_2 \ldots n_k$ av noder
slik at $(n_i, n_{i+1}) \in E$

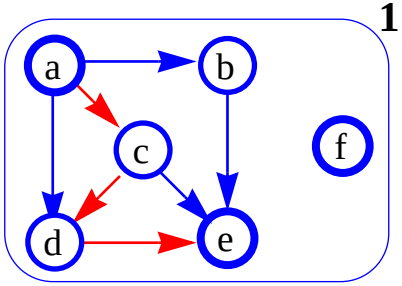
syklus: enkel sti (hver node 1 gang) men $n_1 = n_k$

sammenhengende
graf : det finnes en sti mellom alle par av noder

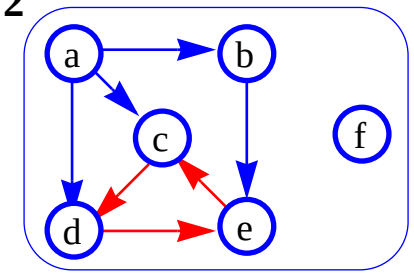
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par $u \rightarrow v$.

1



2



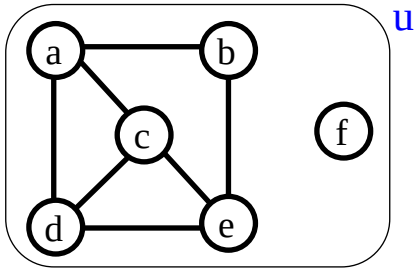
sti : en sekvens av fra-til kanter ... : ade (ikke eda)

rettet syklus: rettet enkel sti ... **2**: cde

Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
$u \rightarrow v$ og $v \rightarrow u$



sti : en sekvens n_1, n_2 ... n_k av noder slik at $(n_i, n_{i+1}) \in E$

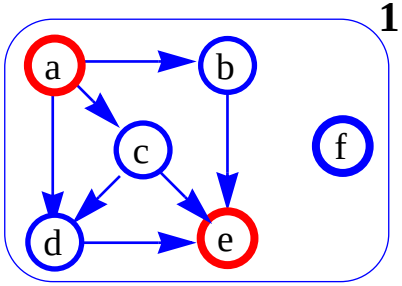
syklus: enkel sti (hver node 1 gang) men $n_1 = n_k$

sammenhengende graf : det finnes en sti mellom alle par av noder

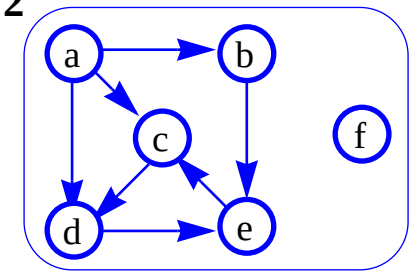
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par $u \rightarrow v$.

1



2



sti : en sekvens av fra-til kanter ... : ade (ikke eda)

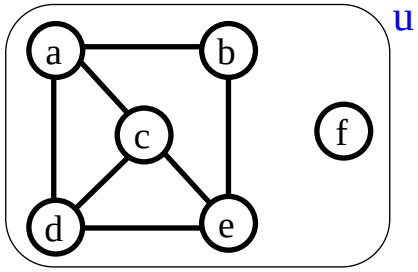
rettet syklus: rettet enkel sti ... **2**: cde

kilde/sluk : en node uten noen inngående / utgående kanter **1**: a/e

Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
$u \rightarrow v$ og $v \rightarrow u$



sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \in E$

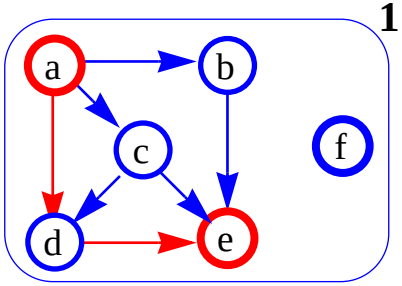
syklus: enkel sti (hver node 1 gang) men $n_1 = n_k$

sammenhengende graf : det finnes en sti mellom alle par av noder

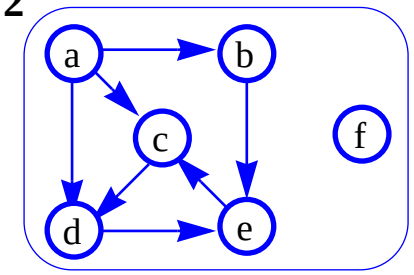
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par $u \rightarrow v$.

1



2



sti : en sekvens av fra-til kanter ... : ade (ikke eda)

rettet syklus: rettet enkel sti ... **2**: cde

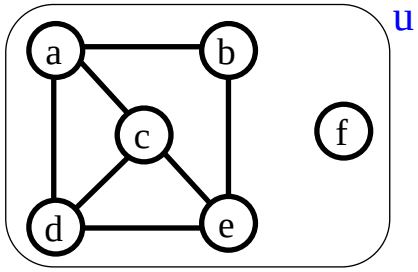
kilde/sluk : en node uten noen inngående / utgående kanter **1**: a/e

oppnåelig (eng: reachable) :en node v kan nåes fra u dersom det finnes en rettet sti fra u til v
e oppnåelig fra a, men ikke omvendt

Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
$u \rightarrow v$ og $v \rightarrow u$



sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \in E$

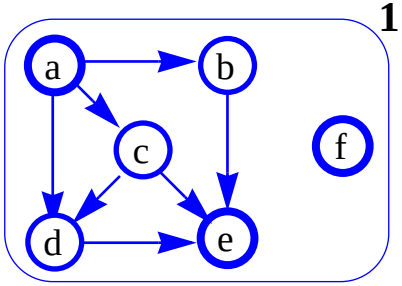
syklus: enkel sti (hver node 1 gang) men $n_1 = n_k$

sammenhengende graf : det finnes en sti mellom alle par av noder

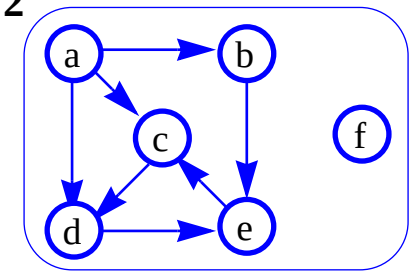
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par $u \rightarrow v$.

1



2



sti : en sekvens av fra-til kanter ... : ade (ikke eda)

rettet syklus: rettet enkel sti ... **2**: cde

kilde/sluk : en node uten noen inngående / utgående kanter **1**: a/e

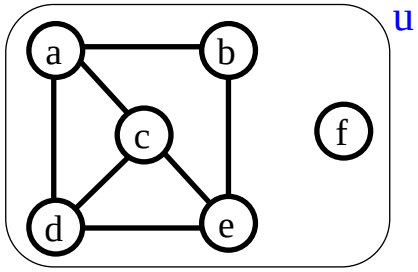
oppnåelig (eng: reachable) :en node v kan nåes fra u dersom det finnes en rettet sti fra u til v
e oppnåelig fra a, men ikke omvendt

sterkt sammenhengende graf : enhvernodeueroppnåeligfraenhverannennodev
hverken **1** eller **2**

Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
u → v og v → u



sti : en sekvens n_1, n_2 ... n_k av noder slik at (n_i,n_{i+1}) ε E

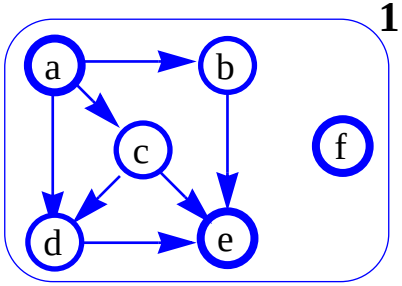
syklus: enkel sti (hver node 1 gang) men $n_1=n_k$

sammenhengende graf : det finnes en sti mellom alle par av noder

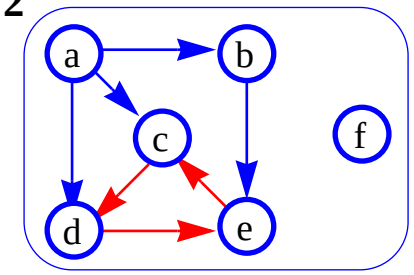
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par u → v.

1



2



sti : en sekvens av fra-til kanter ... : ade (ikke eda)

rettet syklus: rettet enkel sti ... **2**: cde

kilde/sluk : en node uten noen inngående / utgående kanter **1**: a/e

oppnåelig (eng: reachable) :en node v kan nåes fra u dersom det finnes en rettet sti fra u til v
e oppnåelig fra a, men ikke omvendt

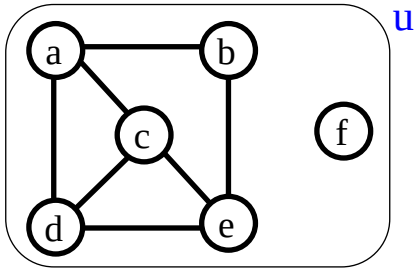
sterkt sammenhengende graf : enhvernodeueroppnåeligfraenhverannennodev hverken **1** eller **2**

DAG : rettet, asyklisk graf – ingen (rettede) sykler
1 er DAG, men ikke **2**

Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
u → v og v → u



sti : en sekvens n_1, n_2 ... n_k av noder slik at (n_i,n_{i+1}) ε E

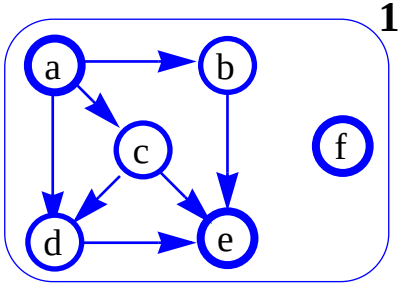
syklus: enkel sti (hver node 1 gang) men $n_1=n_k$

sammenhengende graf : det finnes en sti mellom alle par av noder

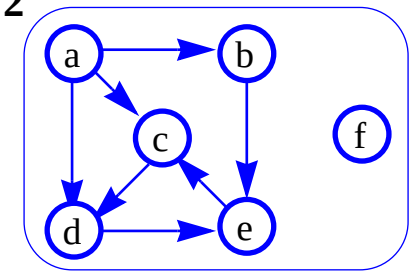
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par u → v.

1



2



sti : en sekvens av fra-til kanter ... : ade (ikke eda)

rettet syklus: rettet enkel sti ... **2**: cde

kilde/sluk : en node uten noen inngående / utgående kanter **1**: a/e

oppnåelig (eng: reachable) :en node v kan nåes fra u dersom det finnes en rettet sti fra u til v
e oppnåelig fra a, men ikke omvendt

sterkt sammenhengende graf : enhvernodeueroppnåeligfraenhverannennodev hverken **1** eller **2**

DAG : rettet, asyklisk graf – ingen (rettede) sykler
1 er DAG, men ikke **2**

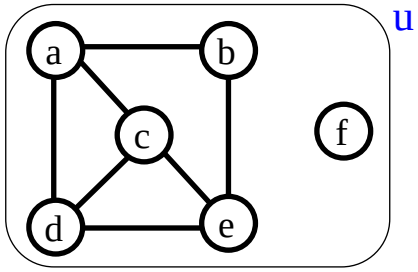
transitiv tillukning

G* **av** G : Har kant u → v hviss G har en sti fra u til v

Graf

URETTET:

en ikke-rettet kant (u,v)
= to rettede kanter
u → v og v → u



sti : en sekvens n_1, n_2 ... n_k av noder slik at (n_i,n_{i+1}) ε E

syklus: enkel sti (hver node 1 gang) men $n_1=n_k$

sammenhengende graf : det finnes en sti mellom alle par av noder

Er v oppnåelig fra u ?

Finn alle v oppnåelige fra u.

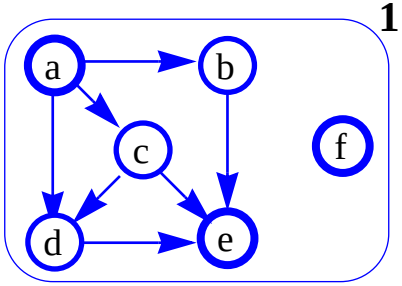
Er G sterkt sammenhengende ?

Er G asyklisk ?

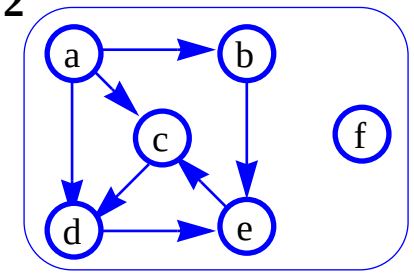
RETTET (DiGraph):

hver kant (u, v) er et ordnet (rettet) par u → v.

1



2



sti : en sekvens av fra-til kanter ... : ade (ikke eda)

rettet syklus: rettet enkel sti ... **2**: cde

kilde/sluk : en node uten noen inngående / utgående kanter **1**: a/e

oppnåelig (eng: reachable) :en node v kan nåes fra u dersom det finnes en rettet sti fra u til v
e oppnåelig fra a, men ikke omvendt

sterkt sammenhengende graf : enhvernodeueroppnåeligfraenhverannennodev hverken **1** eller **2**

DAG : rettet, asyklisk graf – ingen (rettede) sykler
1 er DAG, men ikke **2**

transitiv tillukning

G* **av** G : Har kant u → v hviss G har en sti fra u til v

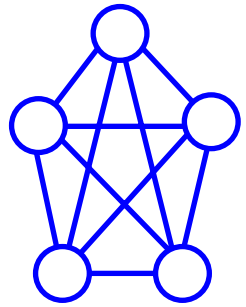
II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

• G er **komplett** **hviss** hver node har (***n*** –1) naboer

dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

n =5
k =10



II. Antall noder og kanter

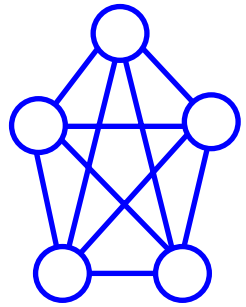
n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

n =5
k =10



II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

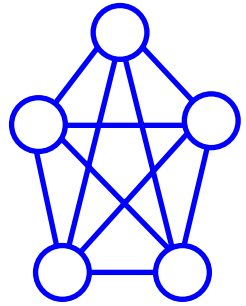
  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

n =5
k =10



II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

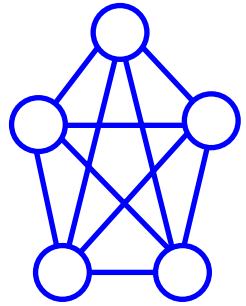
  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

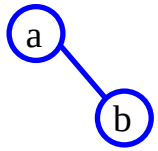
- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

n =5
k =10



n =2
k =1



II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

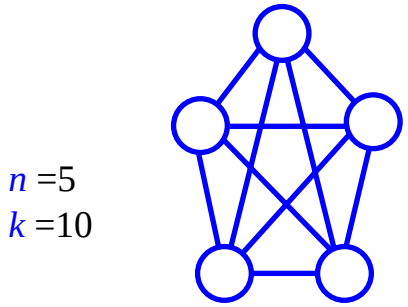
- G er **komplett** **hviss** hver node har (***n*** –1) naboer

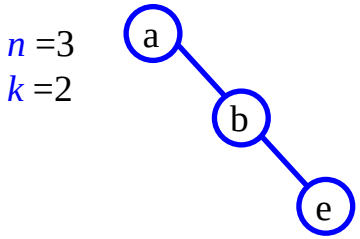
  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?





II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

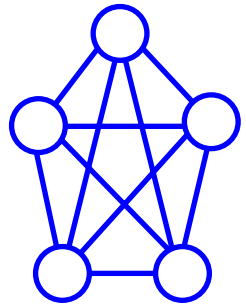
  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

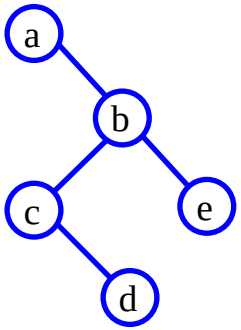
- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

n =5
k =10



n =5
k =4



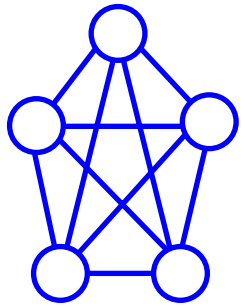
II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

n =5
k =10



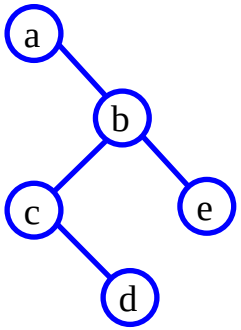
- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

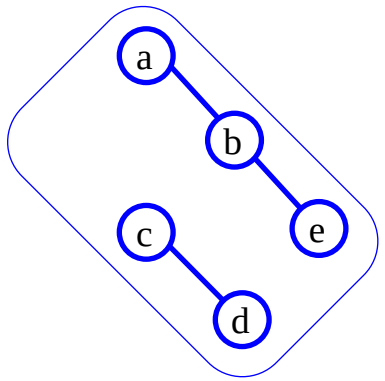
- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

  - fjerning av en vilkårlig kant, gjør treet usammenhengende.

n =5
k =4





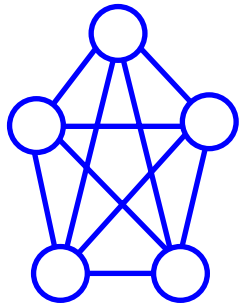
II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

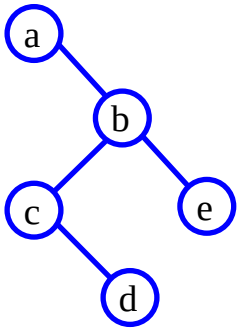
n =5
k =10

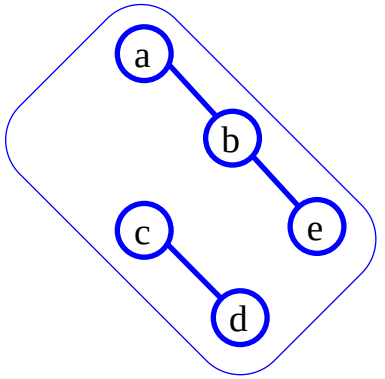


- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

n =5
k =4





  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

  - fjerning av en vilkårlig kant, gjør treet usammenhengende.

  - G trenger ikke å være et tre selv om ***k*** = ***n*** –1 (G kan være usammenhengende)

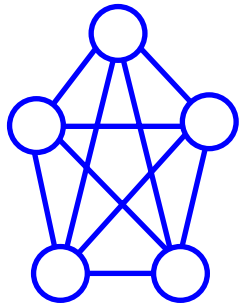
II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

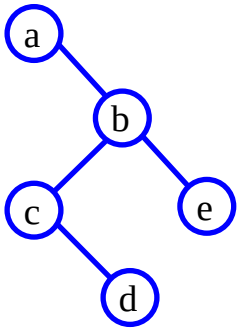
n =5
k =10

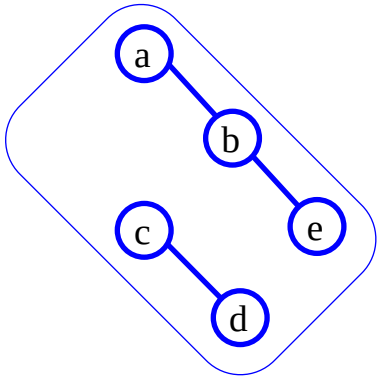


- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

n =5
k =4





  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

  - fjerning av en vilkårlig kant, gjør treet usammenhengende.

  - G trenger ikke å være et tre selv om ***k*** = ***n*** –1 (G kan være usammenhengende)

- **Hvis** ***k*** < ***n*** –1 **så er ikke** G **sammenhengende**

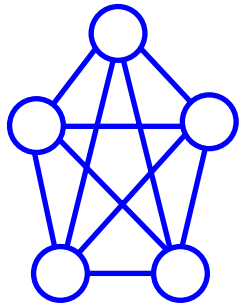
II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

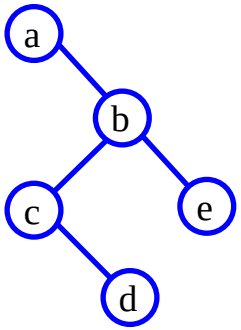
n =5
k =10

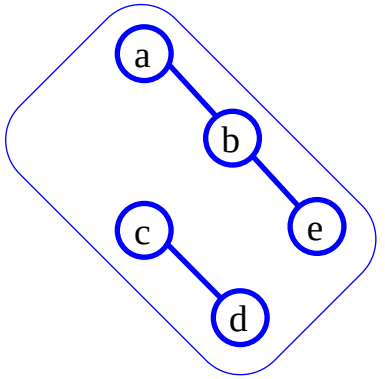


- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

n =5
k =4





  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

  - fjerning av en vilkårlig kant, gjør treet usammenhengende.

  - G trenger ikke å være et tre selv om ***k*** = ***n*** –1 (G kan være usammenhengende)

- **Hvis** ***k*** < ***n*** –1 **så er ikke** G **sammenhengende**
  men ikke omvendt. Hvorfor ikke?

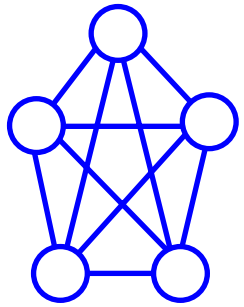
II. Antall noder og kanter

n = # noder i grafen, ***k*** = # kanter i grafen

- G er **komplett** **hviss** hver node har (***n*** –1) naboer

  dvs. ***k*** = 1/2 * $\sum_{v \in V} deg(v)$ = 1/2 * ***n*** * (***n*** –1)

n =5
k =10



- G **ikke komplett** **hviss** ***k*** < 1/2 * ***n*** * (***n*** –1)

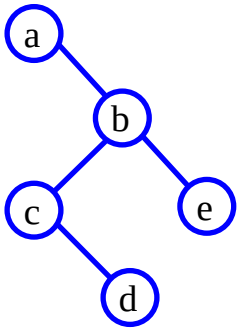
- **Hvis** G **er et tre så er** ***k*** = ***n*** –1

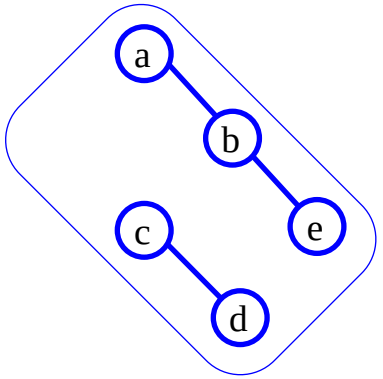
  - dette er det minste antall kanter som kan gi en sammenhengende graf på ***n*** noder. Bevis?

  - fjerning av en vilkårlig kant, gjør treet usammenhengende.

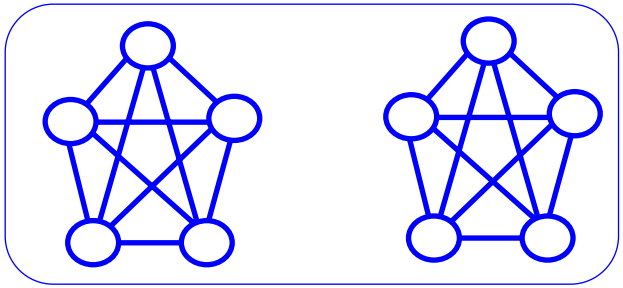
  - G trenger ikke å være et tre selv om ***k*** = ***n*** –1 (G kan være usammenhengende)

n =5
k =4





- **Hvis** ***k*** < ***n*** –1 **så er ikke** G **sammenhengende**
  men ikke omvendt. Hvorfor ikke?

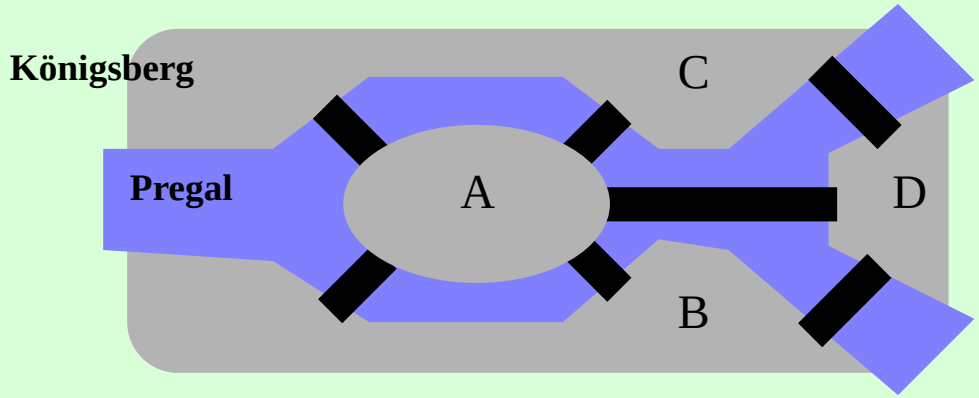


n =10
k =20 > 9

II. *Eulers tur i en vilkårlig graf*

Euler:

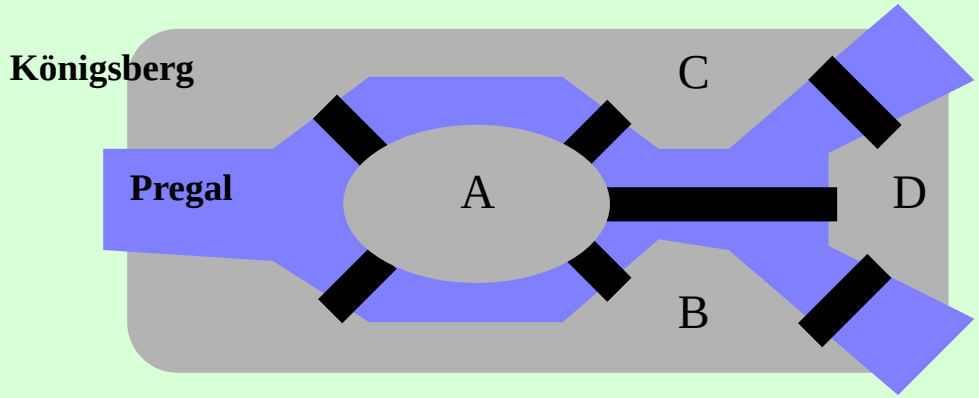
Kan jeg under min aftentur krysse hver bro nøyaktig én gang og returnere til startpunktet?



II. *Eulers tur i en vilkårlig graf*

Euler:

Kan jeg under min aftentur krysse hver bro nøyaktig én gang og returnere til startpunktet?



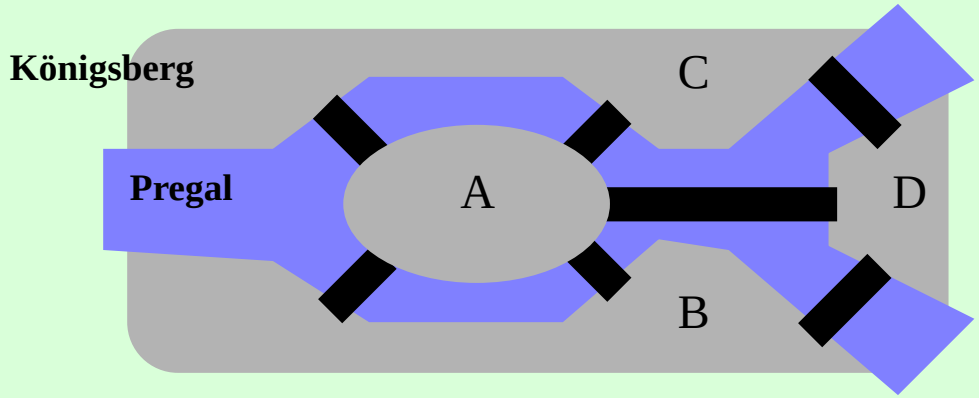
Euler (1736) :

Nei – og jeg kan bevise det v.hj.a. grafer !

II. *Eulers tur i en vilkårlig graf*

Euler:

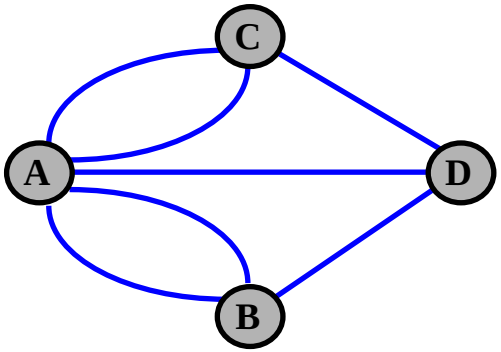
Kan jeg under min aftentur krysse hver bro nøyaktig én gang og returnere til startpunktet?



Euler (1736) :

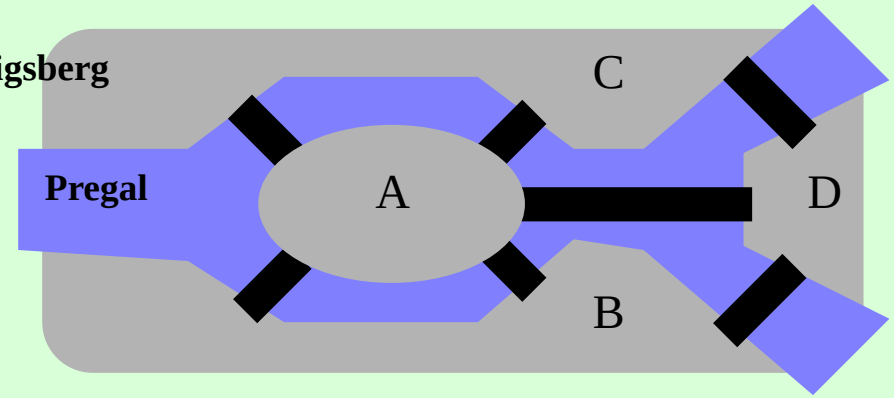
Nei – og jeg kan bevise det v.hj.a. grafer !

Bruk multigrafer (som under), eller la broer være noder med gradtall 2.



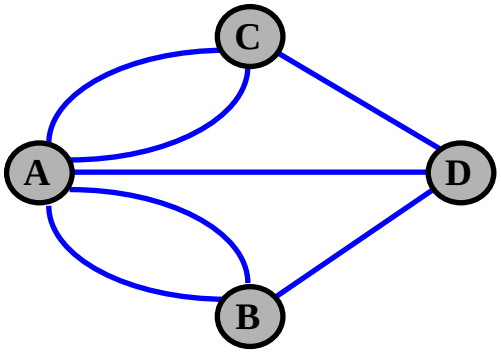
II. *Eulers tur i en vilkårlig graf*

Euler:
Kan jeg under min aftentur krysse hver bro nøyaktig én gang og returnere til startpunktet?



Euler (1736) :
Nei – og jeg kan bevise det v.hj.a. grafer !

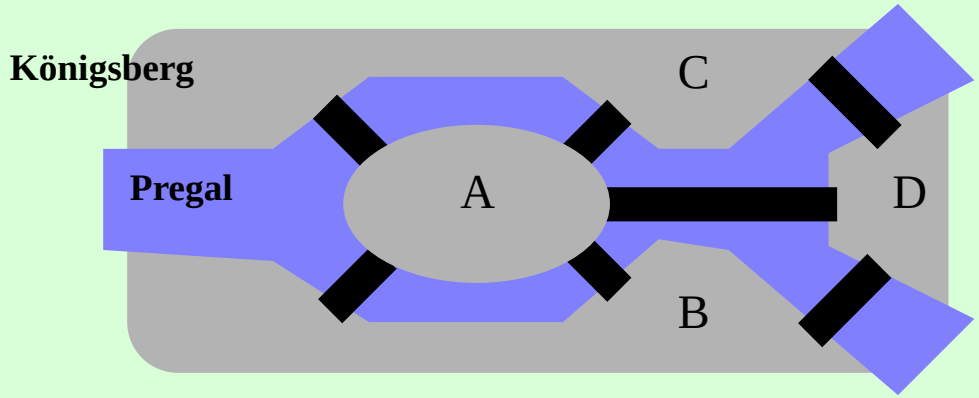
Bruk multigrafer (som under), eller la broer være noder med gradtall 2.



Eulers tur : en sykel som traverserer hver kant i grafen nøyaktig én gang

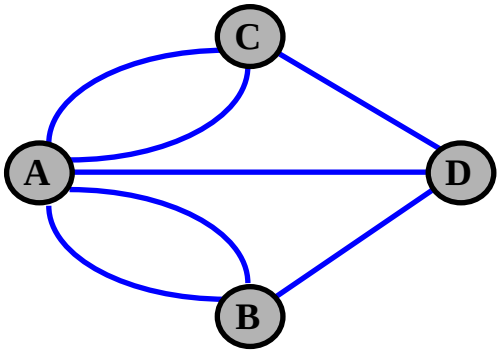
II. *Eulers tur i en vilkårlig graf*

Euler:
Kan jeg under min aftentur krysse hver bro nøyaktig én gang og returnere til startpunktet?



Euler (1736) :
Nei – og jeg kan bevise det v.hj.a. grafer !

Bruk multigrafer (som under), eller la broer være noder med gradtall 2.

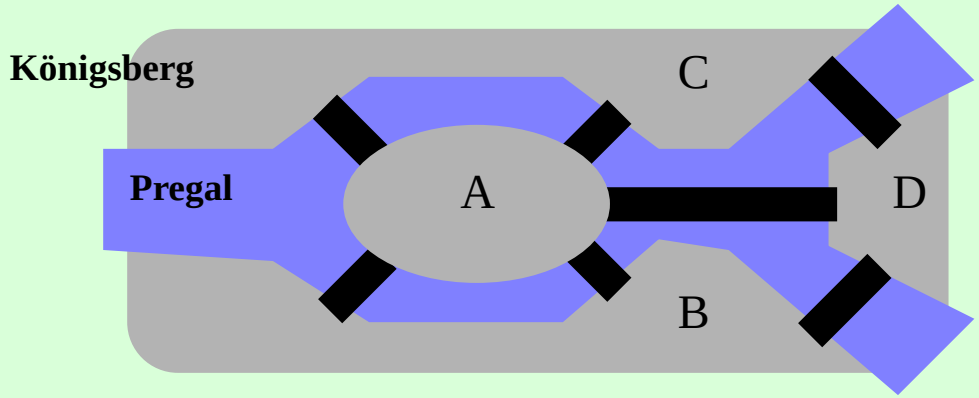


Eulers tur : en sykel som traverserer hver kant i grafen nøyaktig én gang

Eulers teorem : En (sammenhengende) graf G har en Eulers tur hvis og bare hvis
G har 0 eller 2 noder hvor gradtallet er et oddetall. Hvorfor? 'Lett' å finne:**O(n+k)**

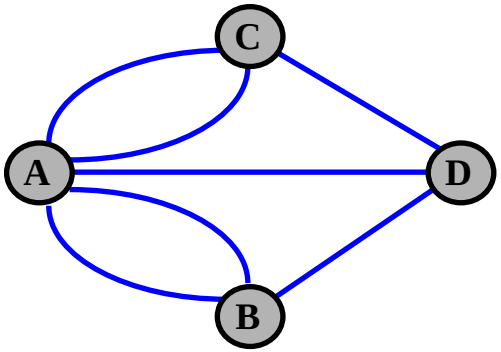
II. *Eulers tur i en vilkårlig graf*

Euler:
Kan jeg under min aftentur krysse hver bro nøyaktig én gang og returnere til startpunktet?



Euler (1736) :
Nei – og jeg kan bevise det v.hj.a. grafer !

Bruk multigrafer (som under), eller la broer være noder med gradtall 2.



Eulers tur : en sykel som traverserer hver kant i grafen nøyaktig én gang

Eulers teorem : En (sammenhengende) graf G har en Eulers tur hvis og bare hvis G har 0 eller 2 noder hvor gradtallet er et oddetall. Hvorfor? 'Lett' å finne:**O(n+k)**

Hamiltonsk sykel : en sykel som traverserer hver node nøyaktig en gang. Vanskelig å finne: **O(n!)**

... **...**

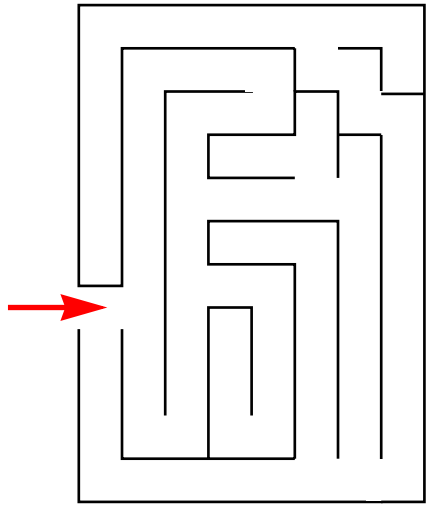
III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

I'm good at killing but how the heck am I going to find it and then get out of there **?**



III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

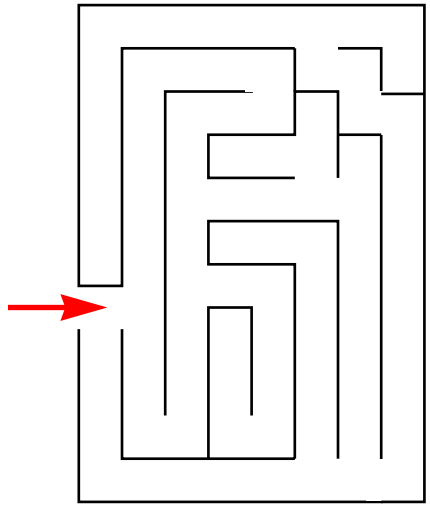
Theseus :

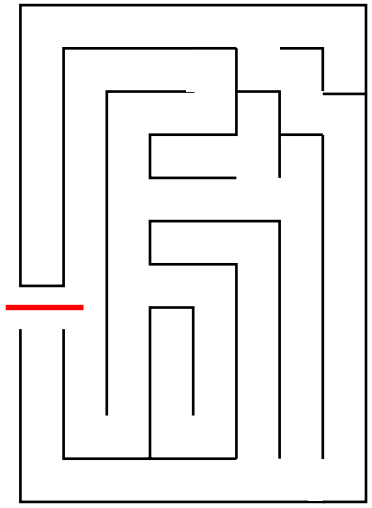
I'm good at killing but how the heck am I going to find it and then get out of there **?**

Ariadne :

(she felt in love with Theseus in the meantime...)

Ta denne tråden og fest den ved inngangen til Labirynten.





III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

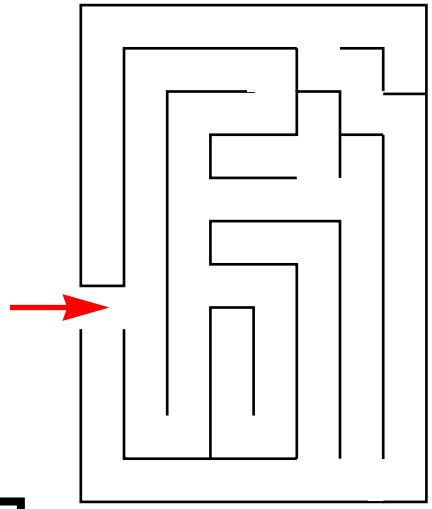
I'm good at killing but how the heck am I going to find it and then get out of there **?**

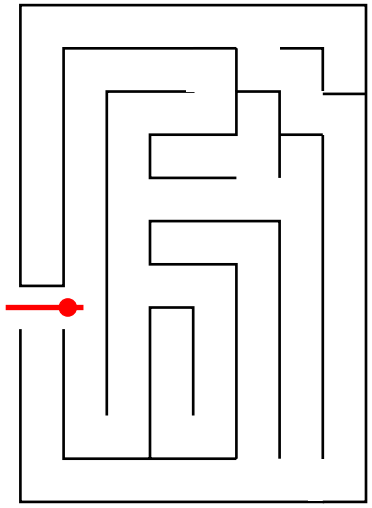
Ariadne :

(she felt in love with Theseus in the meantime...)

Ta denne tråden og fest den ved inngangen til Labirynten.

(**k**) Hver gang du kommer til et nytt kryss **u**, merk **u** '**visited**'.





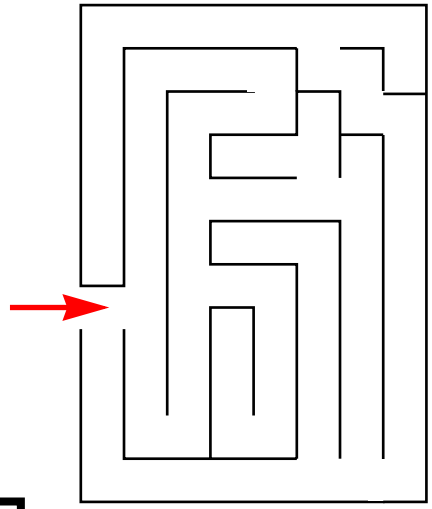
III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

I'm good at killing but how the heck am I going to find it and then get out of there **?**



Ariadne :

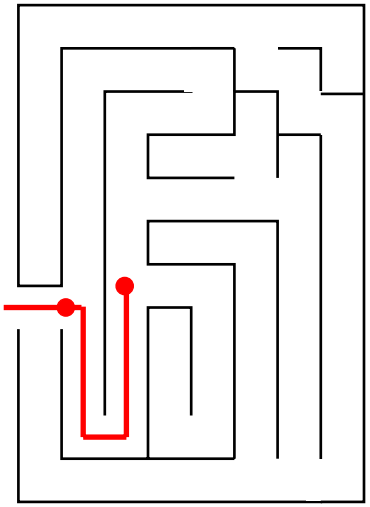
(she felt in love with Theseus in the meantime...)

Ta denne tråden og fest den ved inngangen til Labirynten.

(**k**) Hver gang du kommer til et nytt kryss **u**, merk **u** '**visited**'.

(**r**) Velg en vilkårlig vei – merk den og følg men **dra tråden** etter deg
Når du så kommer til et nytt kryss **v** : dersom det er

– et umerket kryss, **gjenta** det hele (**k**)



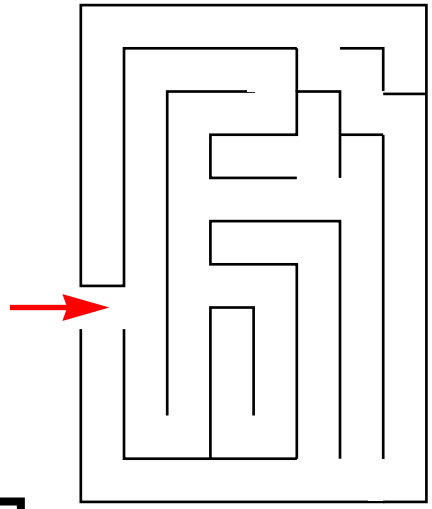
III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

I'm good at killing but how the heck am I going to find it and then get out of there **?**



Ariadne :

(she felt in love with Theseus in the meantime...)

Ta denne tråden og fest den ved inngangen til Labirynten.

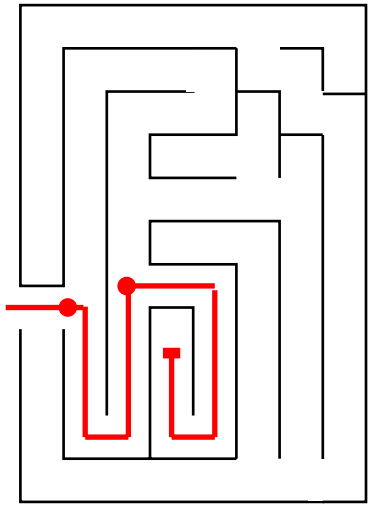
(**k**) Hver gang du kommer til et nytt kryss **u**, merk **u** '**visited**'.

(**r**) Velg en vilkårlig vei – merk den og følg men **dra tråden** etter deg

Når du så kommer til et nytt kryss **v** : dersom det er

– en blind gate, gå tilbake **langs tråden** (**nøst** den igjen)

– et umerket kryss, **gjenta** det hele (**k**)



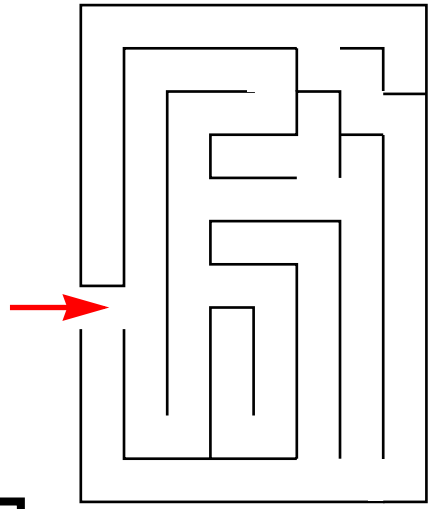
III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

I'm good at killing but how the heck am I going to find it and then get out of there **?**



Ariadne :

(she felt in love with Theseus in the meantime...)

Ta denne tråden og fest den ved inngangen til Labirynten.

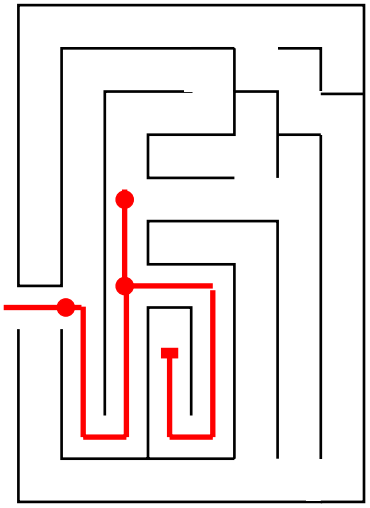
(**k**) Hver gang du kommer til et nytt kryss **u**, merk **u** '**visited**'.

(**r**) Velg en vilkårlig vei – merk den og følg men **dra tråden** etter deg

Når du så kommer til et nytt kryss **v** : dersom det er

– en blind gate, gå tilbake **langs tråden** (**nøst** den igjen)

– et umerket kryss, **gjenta** det hele (**k**)



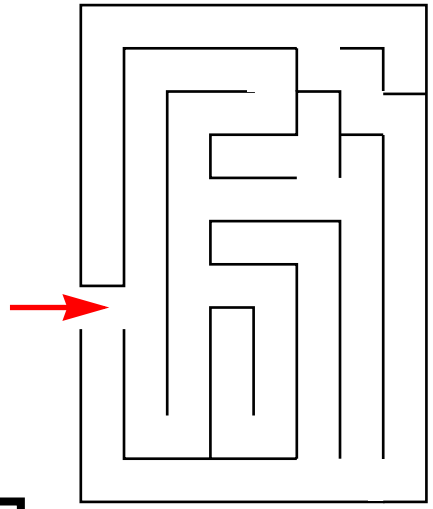
III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

I'm good at killing but how the heck am I going to find it and then get out of there **?**



Ariadne :

(she felt in love with Theseus in the meantime...)

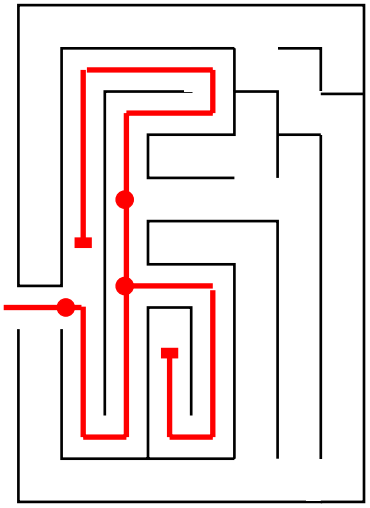
Ta denne tråden og fest den ved inngangen til Labirynten.

(**k**) Hver gang du kommer til et nytt kryss **u**, merk **u** '**visited**'.

(**r**) Velg en vilkårlig vei – merk den og følg men **dra tråden** etter deg

Når du så kommer til et nytt kryss **v** : dersom det er

- en blind gate, gå tilbake **langs tråden** (**nøst** den igjen)
- et kryss merket '**visited**', gå tilbake **langs tråden** (**nøst** den igjen)
- et umerket kryss, **gjenta** det hele (**k**)



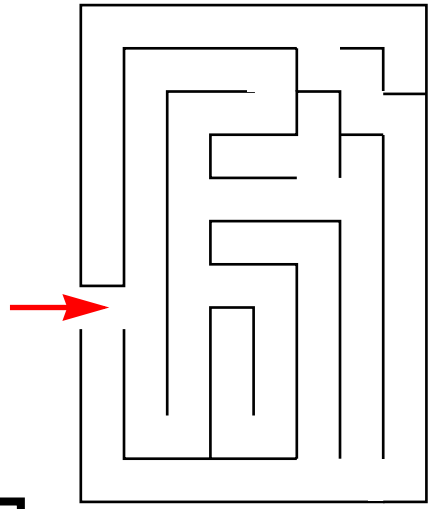
III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

I'm good at killing but how the heck am I going to find it and then get out of there **?**



Ariadne :

(she felt in love with Theseus in the meantime...)

Ta denne tråden og fest den ved inngangen til Labirynten.

(**k**) Hver gang du kommer til et nytt kryss **u**, merk **u** '**visited**'.

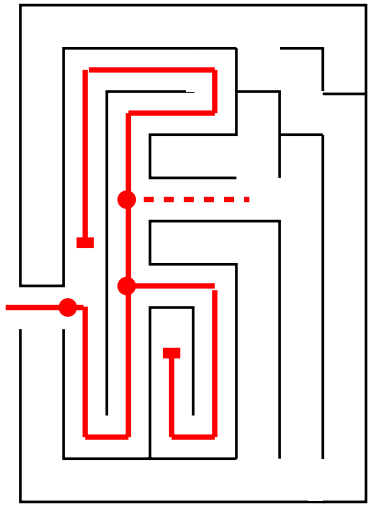
(**r**) Velg en vilkårlig vei – merk den og følg men **dra tråden** etter deg

Når du så kommer til et nytt kryss **v** : dersom det er

- en blind gate, gå tilbake **langs tråden** (**nøst** den igjen)
- et kryss merket '**visited**', gå tilbake **langs tråden** (**nøst** den igjen)
- et umerket kryss, **gjenta** det hele (**k**)

Etter hvert vil alle veier (**r**) fra krysset **u** bli merket

- gå da tilbake **langs tråden** til forrige krysset og fortsett å utforske umerkede veier derfra.



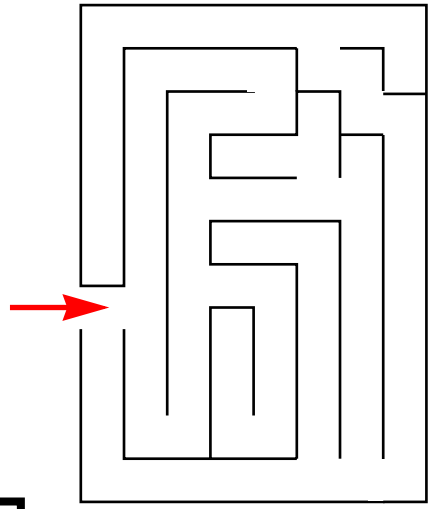
III. *Ariadnes (t)råd*

Minos :

OK, Theseus, but you must first find and kill Minotaur hidding in the Labirynth.

Theseus :

I'm good at killing but how the heck am I going to find it and then get out of there **?**



Ariadne :

(she felt in love with Theseus in the meantime...)

Ta denne tråden og fest den ved inngangen til Labirynten.

(**k**) Hver gang du kommer til et nytt kryss **u**, merk **u** '**visited**'.

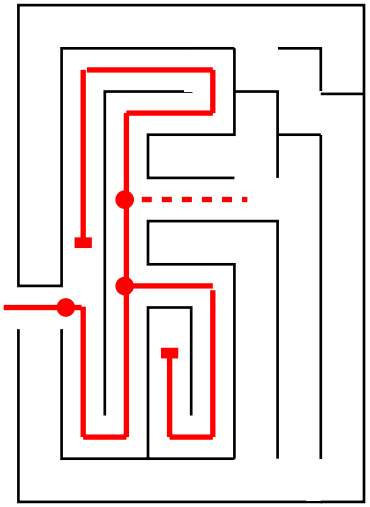
(**r**) Velg en vilkårlig vei – merk den og følg men **dra tråden** etter deg

Når du så kommer til et nytt kryss **v** : dersom det er

- en blind gate, gå tilbake **langs tråden** (**nøst** den igjen)
- et kryss merket '**visited**', gå tilbake **langs tråden** (**nøst** den igjen)
- et umerket kryss, **gjenta** det hele (**k**)

Etter hvert vil alle veier (**r**) fra krysset **u** bli merket

- gå da tilbake **langs tråden** til forrige krysset og fortsett å utforske umerkede veier derfra.



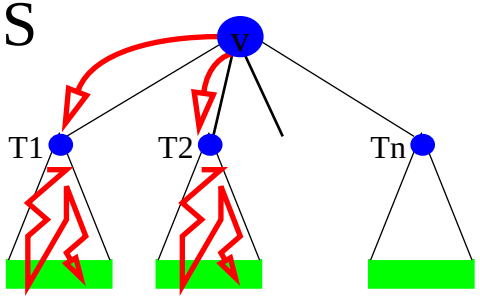
Du vil på denne måten kunne utforske hele labirynten og returnere til inngangen

Chorus :

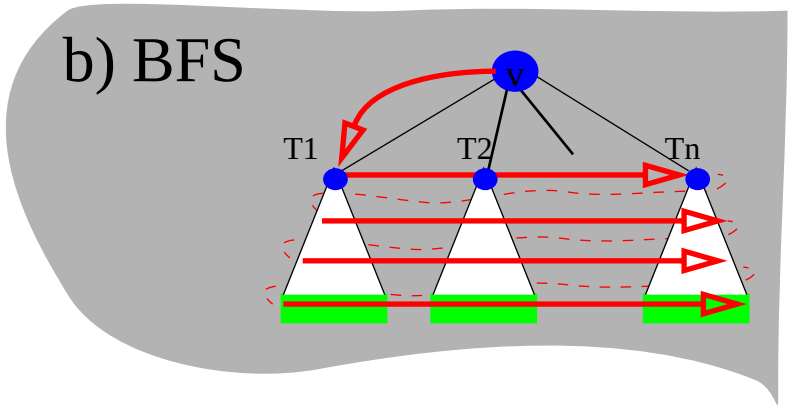
Smart Girl!

III. Graf traversering

a) DFS

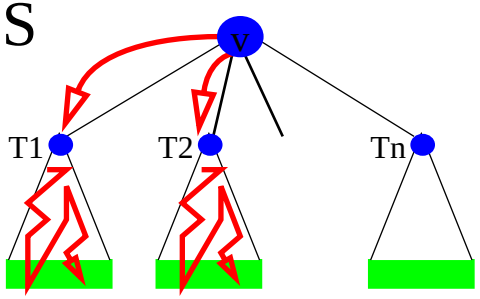


b) BFS

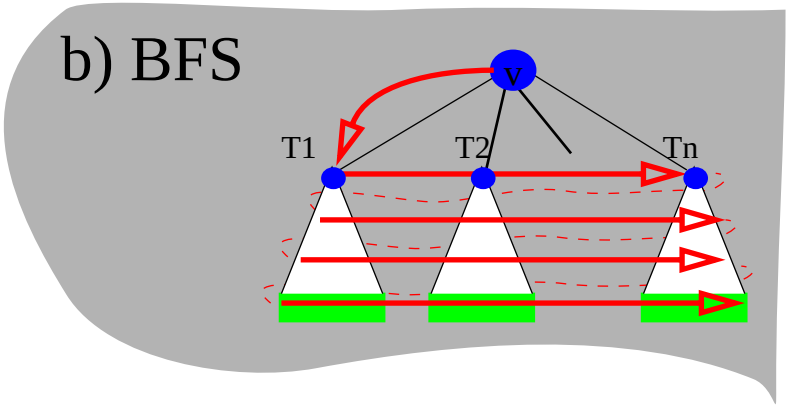


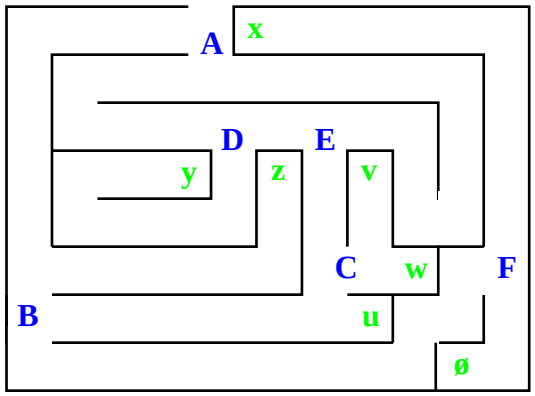
III. Graf traversering

a) DFS



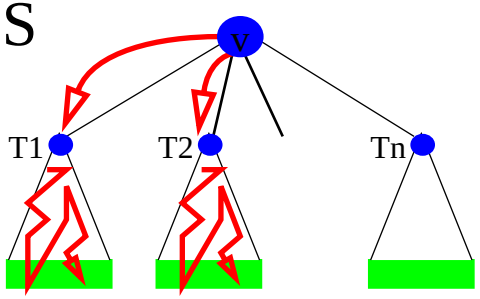
b) BFS



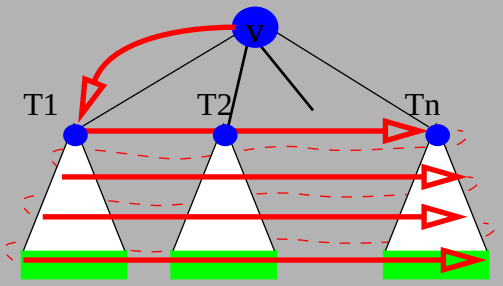


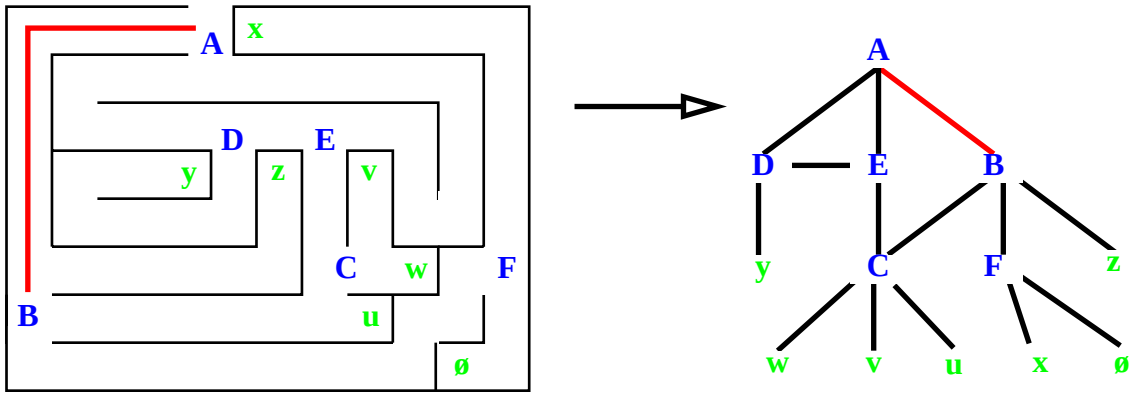
III. Graf traversering

a) DFS



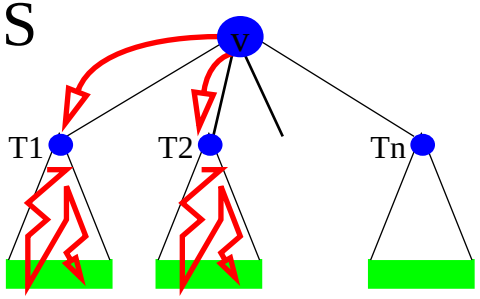
b) BFS



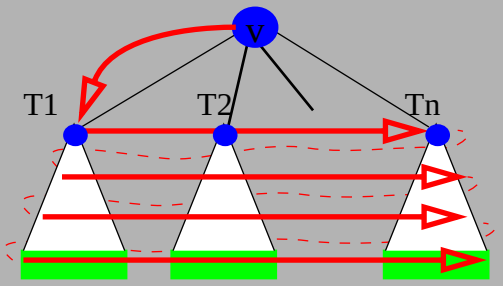


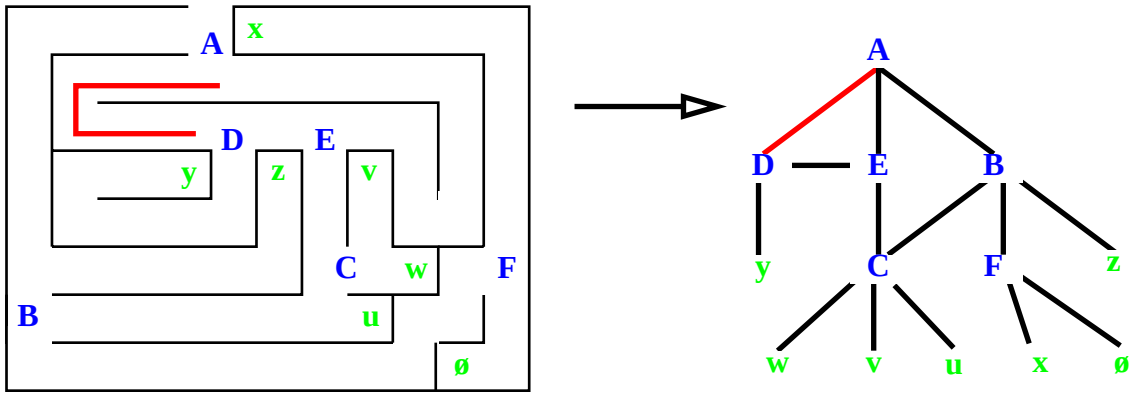
III. Graf traversering

a) DFS



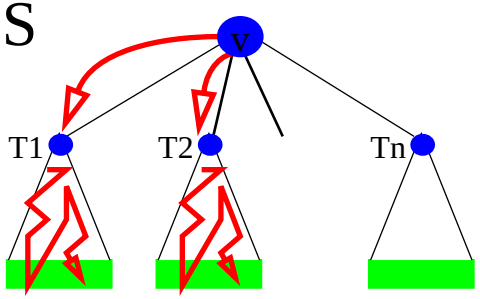
b) BFS



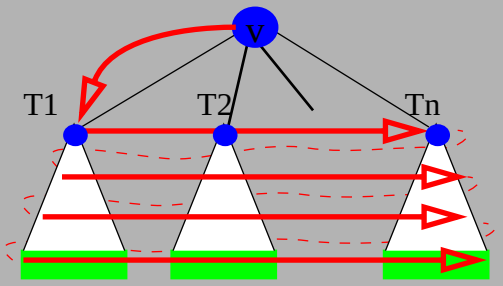


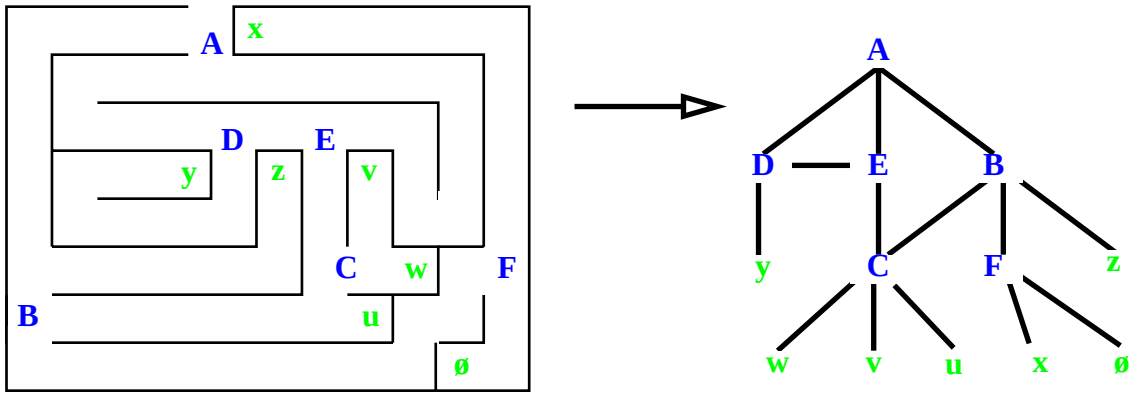
III. Graf traversering

a) DFS



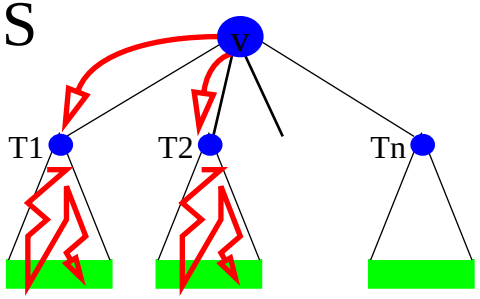
b) BFS



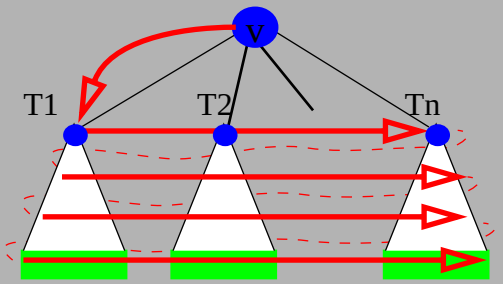


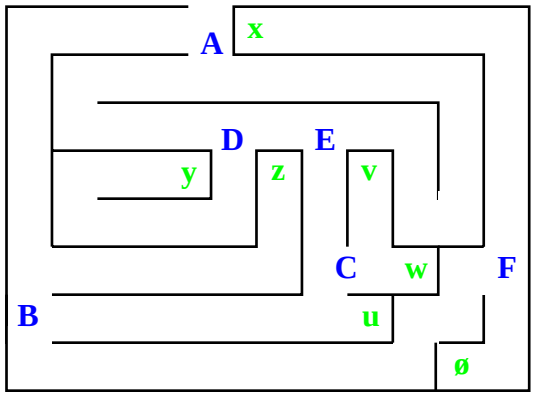
III. Graf traversering

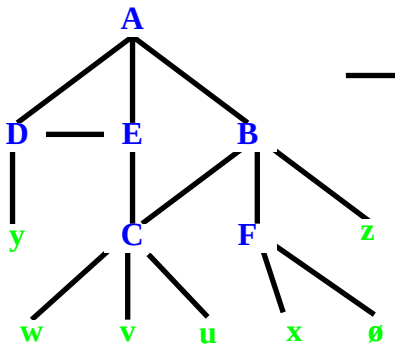
a) DFS



b) BFS



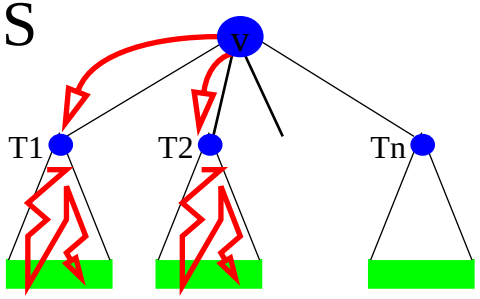




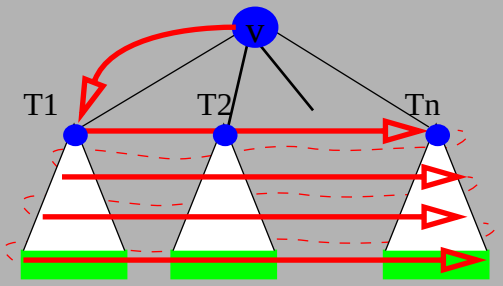

```
DFS(A)
  merk A visited
  for hver kant e = (A, X)
    if ( ! merket X )  // ! e er rød
        // merk–e–rød
        DFS(X)
    // else merk–e–svart
```

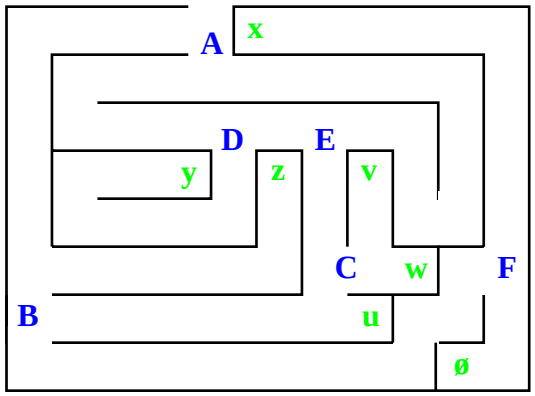
III. Graf traversering

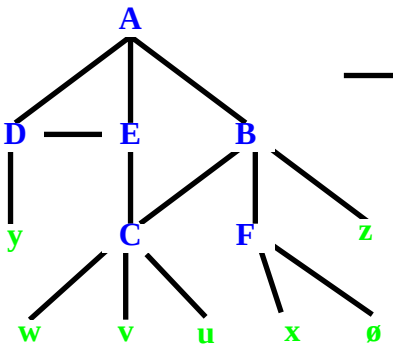
a) DFS



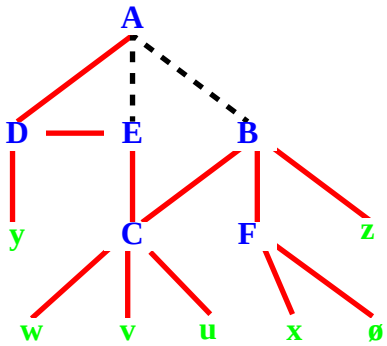
b) BFS







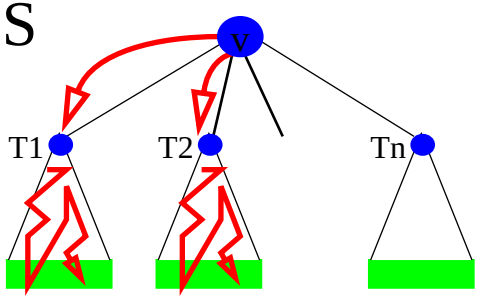

```
DFS(A)
  merk A visited
  for hver kant e = (A, X)
    if ( ! merket X )  // ! e er rød
        // merk–e–rød
        DFS(X)
    // else merk–e–svart
```



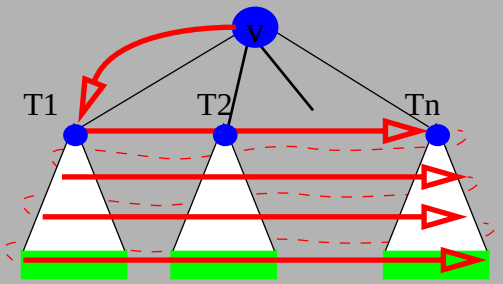
Kanter merket med rød
under DFS traversering gir et
utspennende tre for grafen
– roten til treet kan velges
vilkårlig !

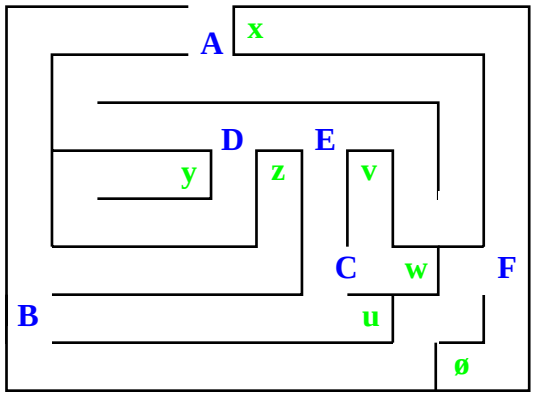
III. Graf traversering

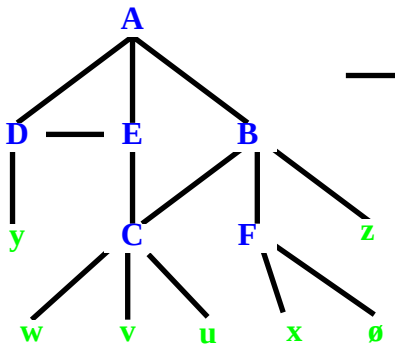
a) DFS



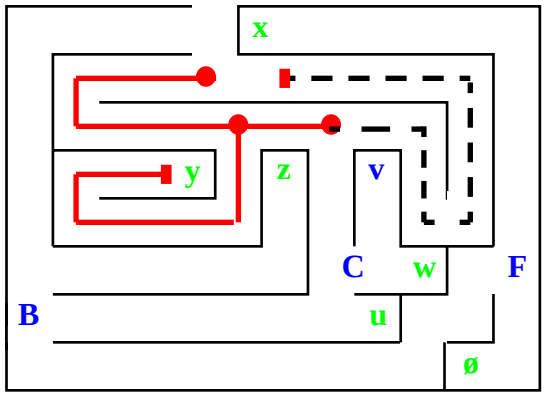
b) BFS

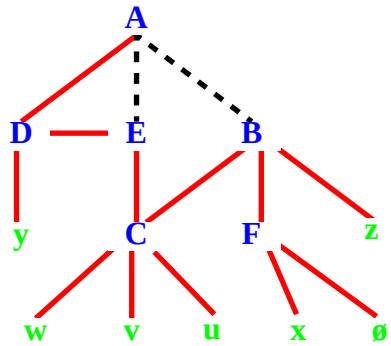







```
DFS(A)
  merk A visited
  for hver kant e = (A, X)
    if ( ! merket X )  // ! e er rød
        // merk–e–rød
        DFS(X)
    // else merk–e–svart
```

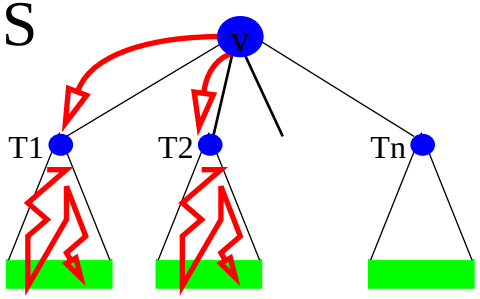




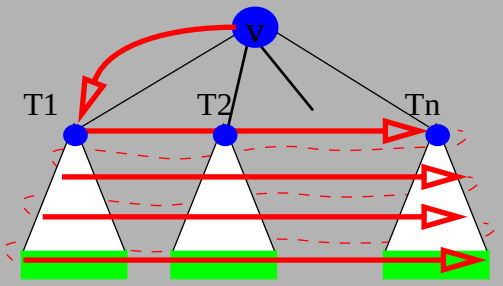
Kanter merket med rød
under DFS traversering gir et
utspennende tre for grafen
– roten til treet kan velges
vilkårlig !

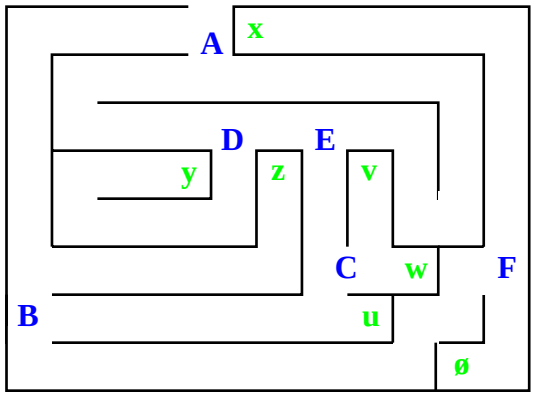
III. Graf traversering

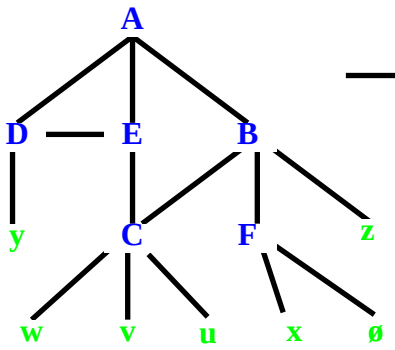
a) DFS



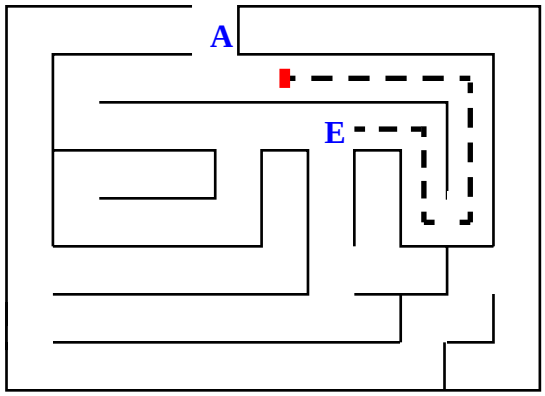
b) BFS

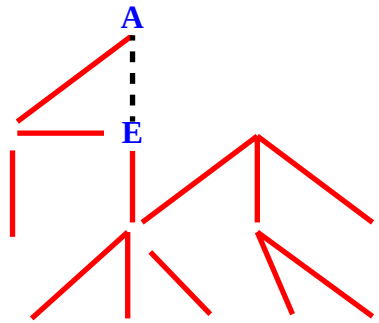







```
DFS(A)
  merk A visited
  for hver kant e = (A, X)
    if ( ! merket X )  // ! e er rød
        // merk–e–rød
        DFS(X)
    // else merk–e–svart
```

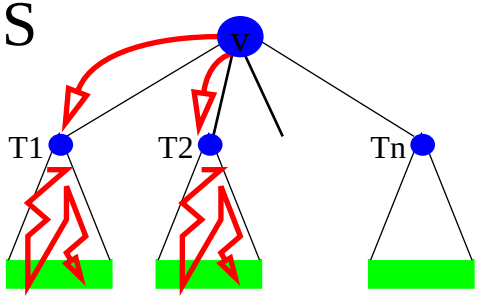




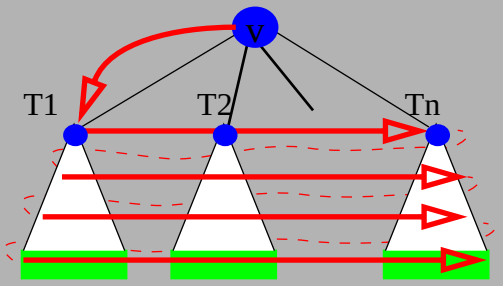
Kanter merket med rød
under DFS traversering gir et
utspennende tre for grafen
– roten til treet kan velges
vilkårlig !

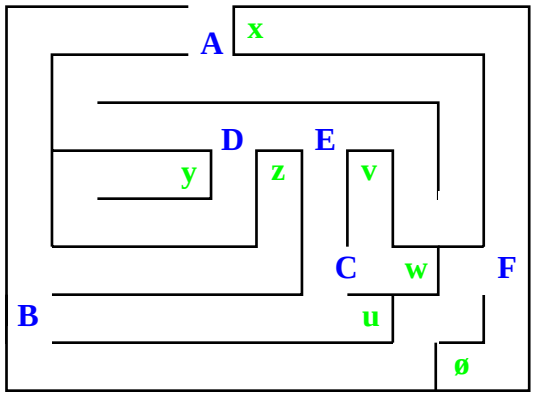
III. Graf traversering

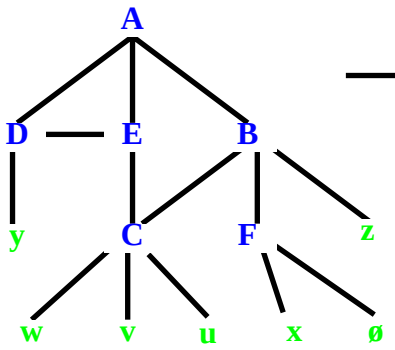
a) DFS



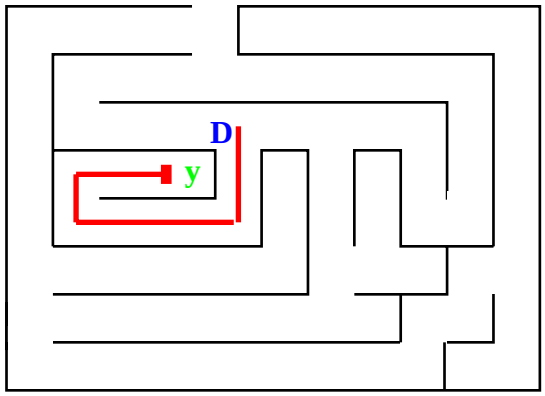
b) BFS

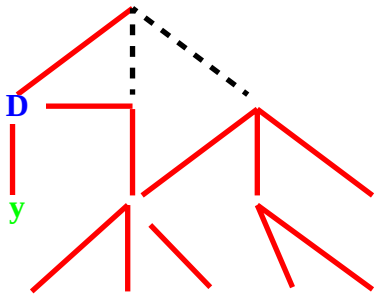







```
DFS(A)
  merk A visited
  for hver kant e = (A, X)
    if ( ! merket X )  // ! e er rød
        // merk–e–rød
        DFS(X)
    // else merk–e–svart
```

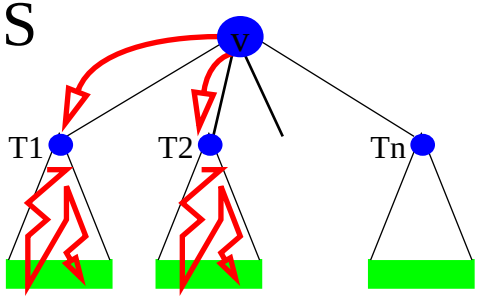




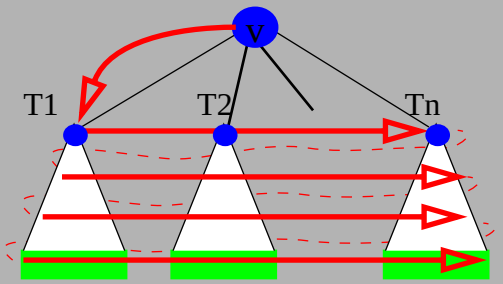
Kanter merket med rød
under DFS traversering gir et
utspennende tre for grafen
– roten til treet kan velges
vilkårlig !

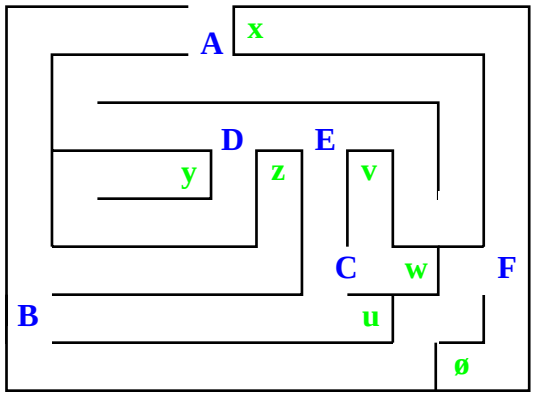
III. Graf traversering

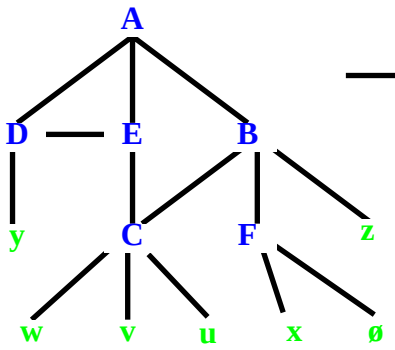
a) DFS



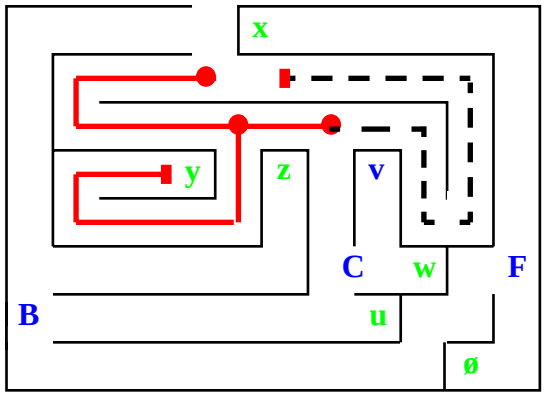
b) BFS

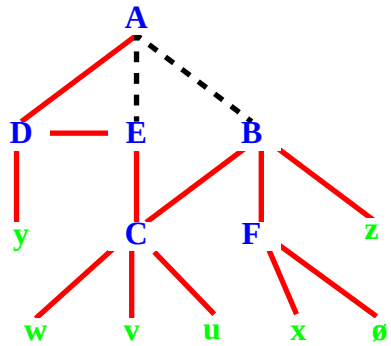







```
DFS(A)
  merk A visited
  for hver kant e = (A, X)
    if ( ! merket X )  // ! e er rød
        // merk–e–rød
        DFS(X)
    // else merk–e–svart
```





Kanter merket med rød
under DFS traversering gir et
utspennende tre for grafen
– roten til treet kan velges
vilkårlig !

III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

a) besøker *alle* noder i en *sammenhengende komponent* til s

Begrunnelse :

a) Anta, kontrapositivt, at det finnes en ubesøkt node v.

Siden komponenten er sammenhengende, finnes det en sti fra s til enhver node,

så anta at v er den første ubesøkte noden på en sti fra s:

- Siden v er den første slike, finnes det en nabo node u ("like før") som ble besøkt
- Men da – mens vi besøkte u – måtte vi også ha sett på kanten (u,v) og, siden v ikke var merket, måtte den ha blitt besøkt.

III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)

III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

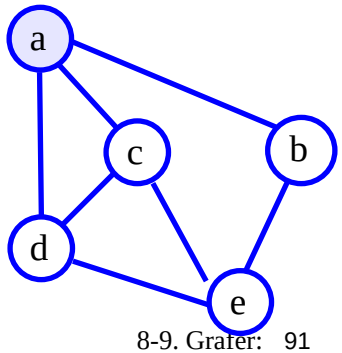
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

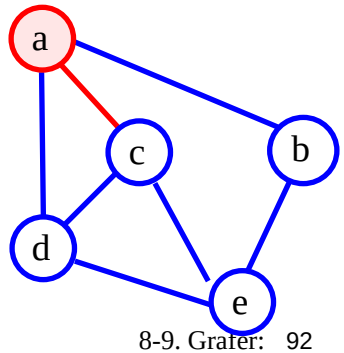
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

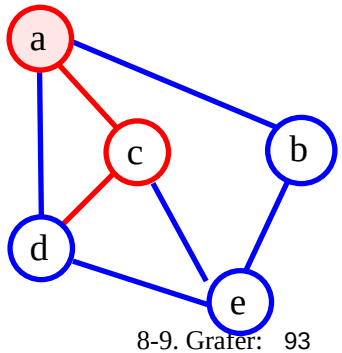
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

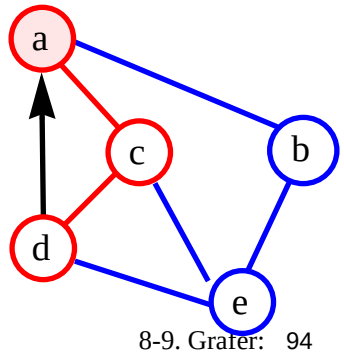
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

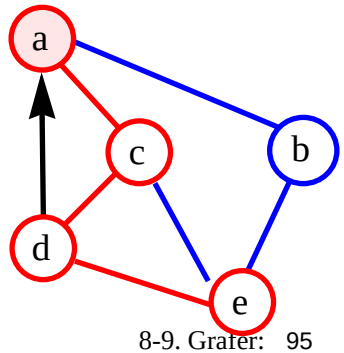
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

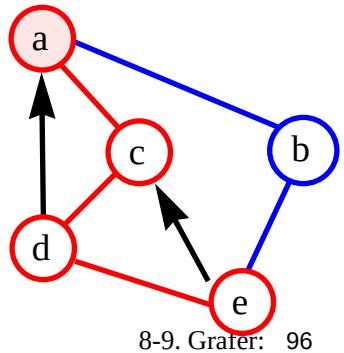
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

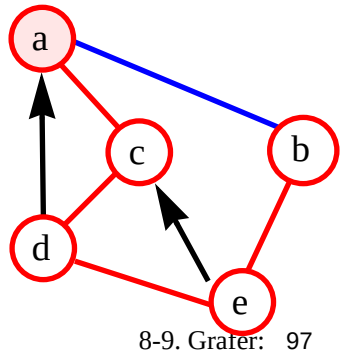
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

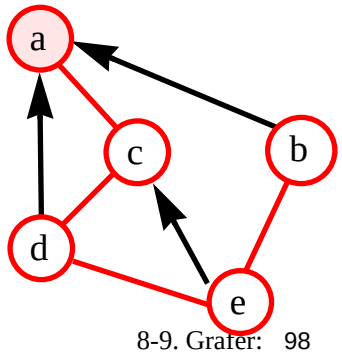
a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

12.12 DFS traversering av en ikke-rettet graf G fra en node s:

a) besøker *alle* noder i en *sammenhengende komponent* til s

b) merkete kanter gir et *utspennende tre*, DFS tre, for denne komponenten til s

Begrunnelse :

a) Anta, kontrapositivt, at det finnes en ubesøkt node v.

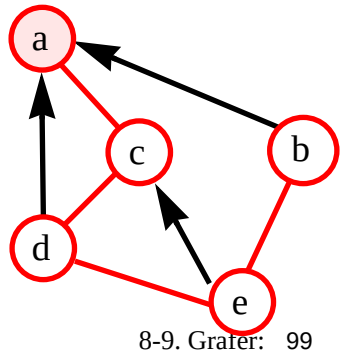
Siden komponenten er sammenhengende, finnes det en sti fra s til enhver node,

så anta at v er den første ubesøkte noden på en sti fra s:

- Siden v er den første slike, finnes det en nabo node u ("like før") som ble besøkt
- Men da – mens vi besøkte u – måtte vi også ha sett på kanten (u,v) og, siden v ikke var merket, måtte den ha blitt besøkt.

b) Vi merker kanter (u,v) kun når vi går til endenoden v for første gang

- Derfor danner vi aldri en sykel – vi fåren asyklisk graf, dvs. et tre
- Treet er utspennende fordi alle komponentens noder er med (a)



III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

Kjøretid av DFS:

DFS kalles 1 gang for hver node og ser hver kant 2 ganger :
$O(n_s+k_s)$ hvis

- gitt en kant, kan man aksessere dens ende-noder i $O(1)$
- merking av noder/kanter og sjekking om de er merket tar $O(1)$
- for hver node v, kan alle dens kanter akssesere 1 gang i $O(k(v))$

III.a) DFS traversering av grafer

```
DFS(s)
  merk s visited
  for hver kant e = (s,u)
      if ( ! merket–u )
          // merk–e–rød
          DFS(u)
```

Kjøretid av DFS:

DFS kalles 1 gang for hver node og ser hver kant 2 ganger :
$O(n_s+k_s)$ hvis

- gitt en kant, kan man aksessere dens ende-noder i $O(1)$
- merking av noder/kanter og sjekking om de er merket tar $O(1)$
- for hver node v, kan alle dens kanter akssesere 1 gang i $O(k(v))$

DFS kan brukes for å lage $O(n+k)$ algoritmer for å :

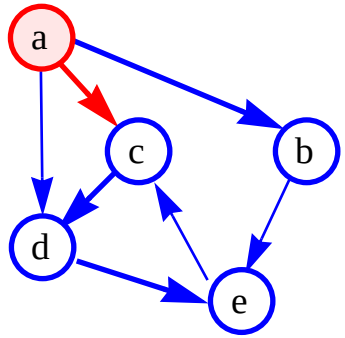
- sjekke om G er sammenhengende og finne sammenhengende komponenter
- finne utspennende tre for G
- sjekke om det finnes en sti mellom to noder;
- se om G inneholder sykler

III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```

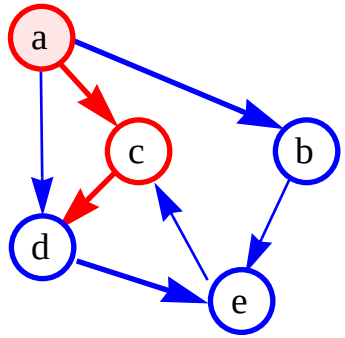
III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```



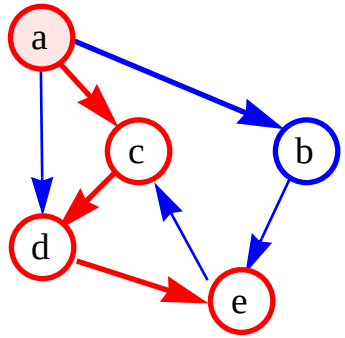
III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```



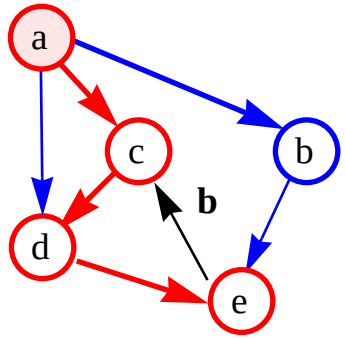
III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```



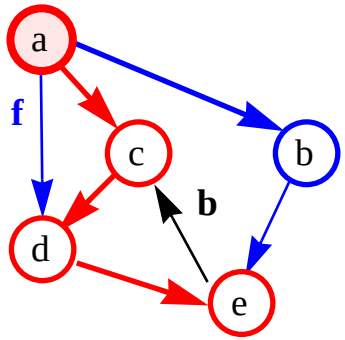
III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```



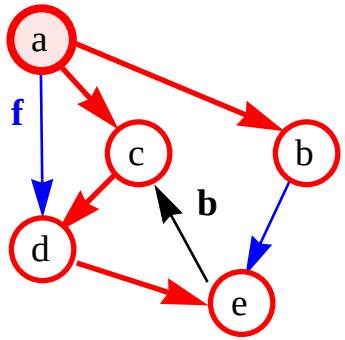
III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```



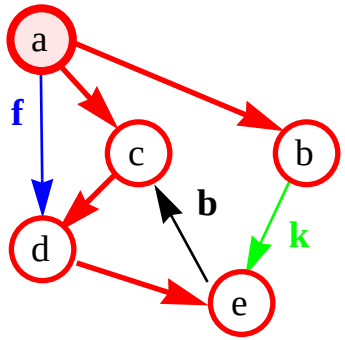
III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```



III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```

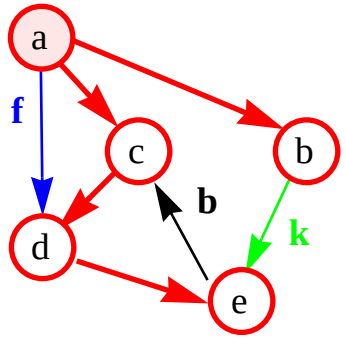


III.a) DFS på en rettet graf

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```

– Traverserte kanter som ikke er med i DFS-treet kan deles i tre grupper:

- **fram**-kanter fra v til en etterfølger node i DFS-treet
- **tilbake**-kanter fra v til en forgjenger node i DFS-treet
- **kryss**-kanter fra v til en urelatert node i DFS-treet



III.a) DFS på en rettet graf

12.16 (12.12) DFS traversering av en rettet graf G fra en node s:

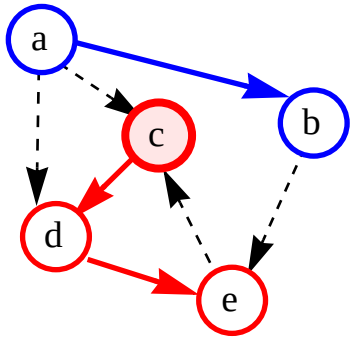
a) besøker alle noder *oppnåelige* fra s

b) gir et utspennende tre, DFS-treet, for delgrafen oppnåelig fra s

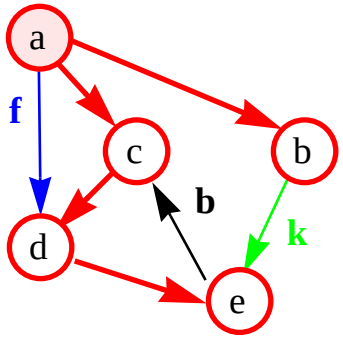
– Traverserte kanter som ikke er med i DFS-treet kan deles i tre grupper:

- **fram**-kanter fra v til en etterfølger node i DFS-treet
- **tilbake**-kanter fra v til en forgjenger node i DFS-treet
- **kryss**-kanter fra v til en urelatert node i DFS-treet

– Iterert DFS: 'for hver node v utfør DFS(v)' kan gi en skog:




```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```



III.a) DFS på en rettet graf

12.16 (12.12) DFS traversering av en rettet graf G fra en node s:

a) besøker alle noder *oppnåelige* fra s

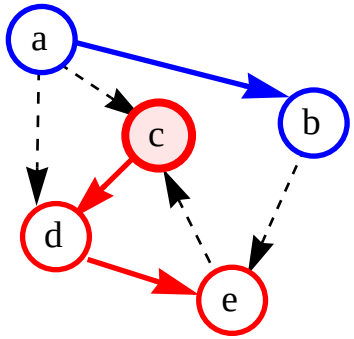
b) gir et utspennende tre, DFS-treet, for delgrafen oppnåelig fra s

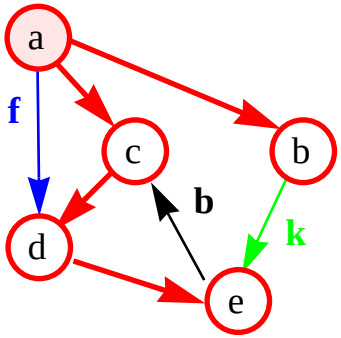
```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```

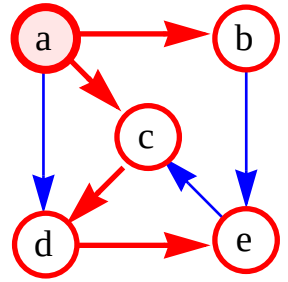
– Traverserte kanter som ikke er med i DFS-treet kan deles i tre grupper:

- **fram**-kanter fra v til en etterfølger node i DFS-treet
- **tilbake**-kanter fra v til en forgjenger node i DFS-treet
- **kryss**-kanter fra v til en urelatert node i DFS-treet

– Iterert DFS: 'for hver node v utfør DFS(v)' kan gi en skog:







kompleksitet	kant-liste	nabo-liste	nabo-matrise
outIncidentEdges(v)	O(**k**)	O(**deg**)	O(**n**)

III.a) DFS på en rettet graf

12.16 (12.12) DFS traversering av en rettet graf G fra en node s:

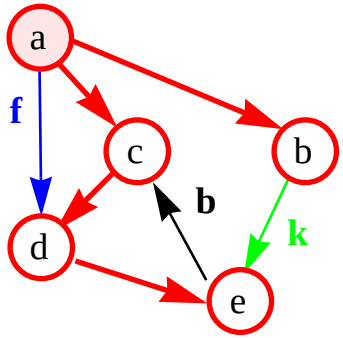
a) besøker alle noder *oppnåelige* fra s

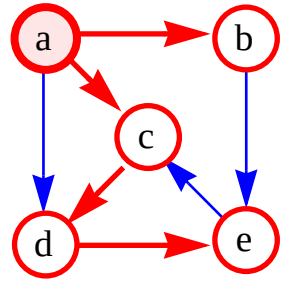
b) gir et utspennende tre, DFS-treet, for delgrafen oppnåelig fra s

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```

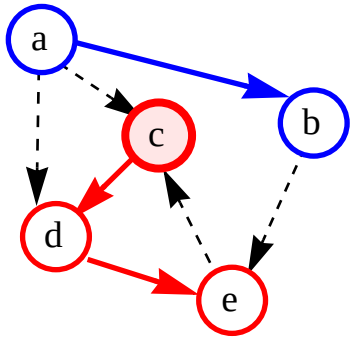
– Traverserte kanter som ikke er med i DFS-treet kan deles i tre grupper:

- **fram**-kanter fra v til en etterfølger node i DFS-treet
- **tilbake**-kanter fra v til en forgjenger node i DFS-treet
- **kryss**-kanter fra v til en urelatert node i DFS-treet





– Iterert DFS: 'for hver node v utfør DFS(v)' kan gi en skog:



kompleksitet	kant-liste	nabo-liste	nabo-matrise
outIncidentEdges(v)	O(**k**)	O(**deg**)	O(**n**)
DFS	**O(n * k)**	**O(n + k)**	**O(n * n)**
siden k=O(n*n) får vi	O($\mathbf{n^3}$)	**O(n^2)**	**O(n^2)**

III.a) DFS på en rettet graf

12.16 (12.12) DFS traversering av en rettet graf G fra en node s:

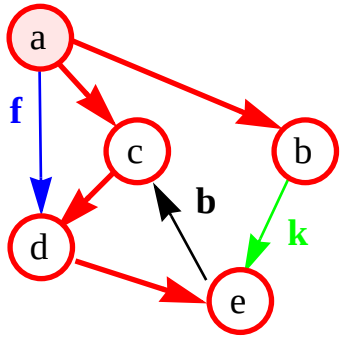
a) besøker alle noder *oppnåelige* fra s

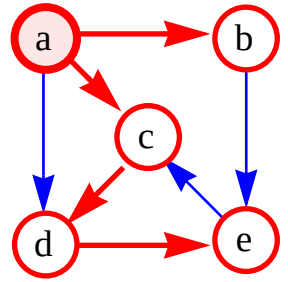
b) gir et utspennende tre, DFS-treet, for delgrafen oppnåelig fra s

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```

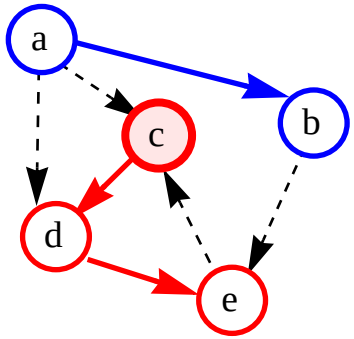
– Traverserte kanter som ikke er med i DFS-treet kan deles i tre grupper:

- **fram**-kanter fra v til en etterfølger node i DFS-treet
- **tilbake**-kanter fra v til en forgjenger node i DFS-treet
- **kryss**-kanter fra v til en urelatert node i DFS-treet





– Iterert DFS: 'for hver node v utfør DFS(v)' kan gi en skog:



kompleksitet	kant-liste	nabo-liste	nabo-matrise
outIncidentEdges(v)	O(**k**)	O(**deg**)	O(**n**)
DFS	**O(n * k)**	**O(n + k)**	**O(n * n)**
siden k=O(n*n) får vi	O($\mathbf{n^{3)}}$	**O(n^2)**	**O(n^2)**

DFS på rettet graf gir opphav til *O***(n+k)** algoritme for å :

- finne alle noder oppnåelig fra en gitt node

Iterert DFS gir *O***(n(n+k))** algoritmer for å :

- avgjøre om G er sterkt sammenhengende; (mulig også i O(n+k))
- lage transitiv tilluking G* av G

III.a) DFS på en rettet graf

12.16 (12.12) DFS traversering av en rettet graf G fra en node s:

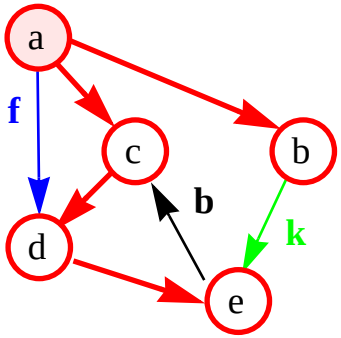
a) besøker alle noder *oppnåelige* fra s

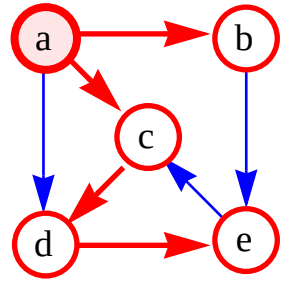
b) gir et utspennende tre, DFS-treet, for delgrafen oppnåelig fra s

```
DFS(s)                    // opptil n rekursive kall
  merk–s visited
  for hver kant e ε outIncidentEdges(s)
      v = opposite(s,e)
      if ( ! merket(v) )
          // merk e rød
          ..... DFS(v)
```

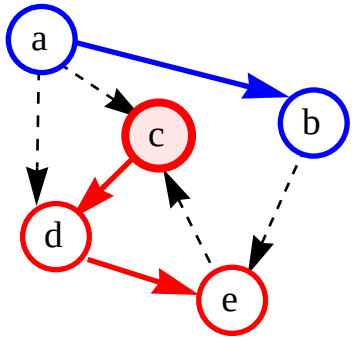
– Traverserte kanter som ikke er med i DFS-treet kan deles i tre grupper:

- **fram**-kanter fra v til en etterfølger node i DFS-treet
- **tilbake**-kanter fra v til en forgjenger node i DFS-treet
- **kryss**-kanter fra v til en urelatert node i DFS-treet





– Iterert DFS: 'for hver node v utfør DFS(v)' kan gi en skog:



– BFS for rettede grafer har tilsvarende egenskaper til BFS for ikke-rettede grafer (etterlater kun tilbake- og kryss-kanter)

kompleksitet	kant-liste	nabo-liste	nabo-matrise
outIncidentEdges(v)	O(**k**)	O(**deg**)	O(**n**)
DFS	**O(n * k)**	**O(n + k)**	**O(n * n)**
siden k=O(n*n) får vi	O($\mathbf{n^{3}}$)	**O(n^2)**	**O(n^2)**

DFS på rettet graf gir opphav til ***O*(n+k)** algoritme for å :

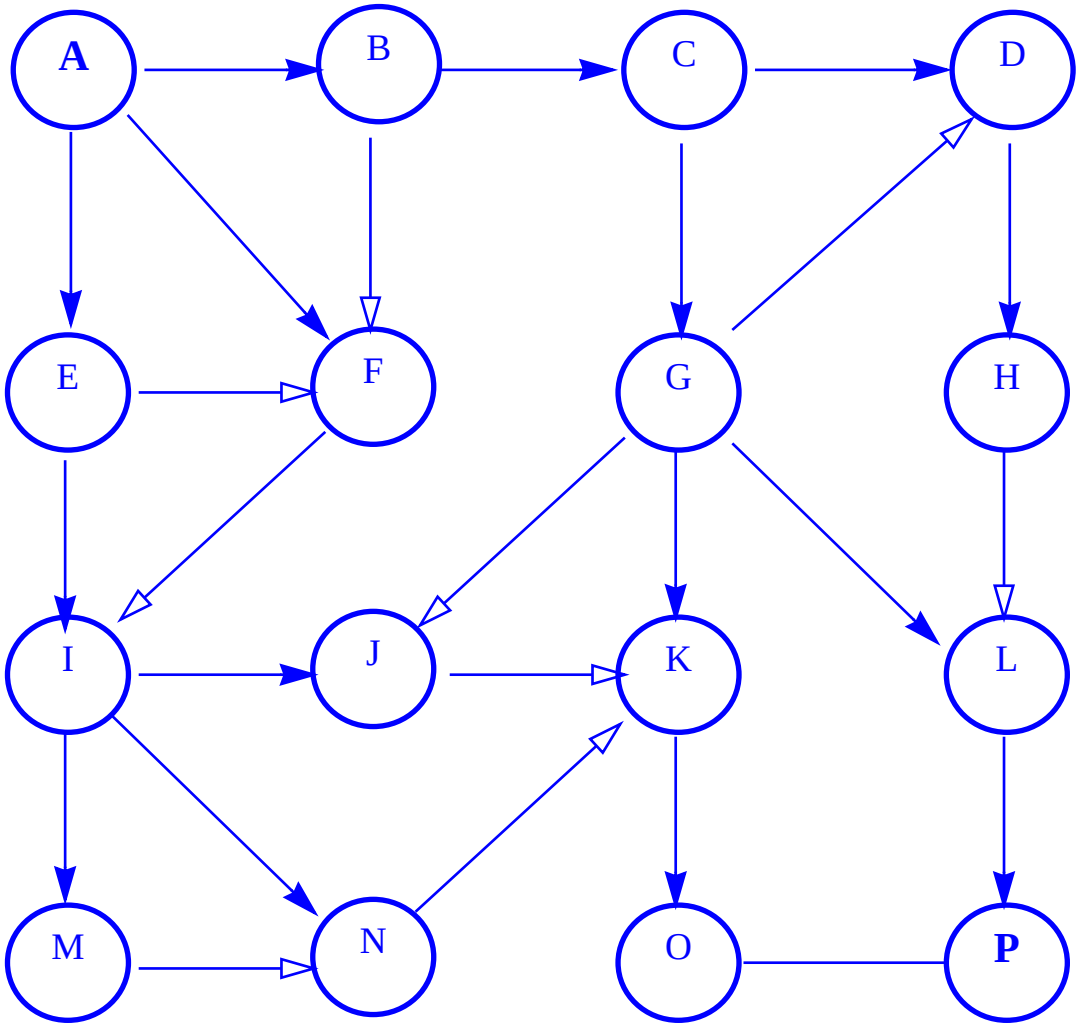
- finne alle noder oppnåelig fra en gitt node

Iterert DFS gir ***O*(n(n+k))** algoritmer for å :

- avgjøre om G er sterkt sammenhengende; (mulig også i O(n+m))
- lage transitiv tilluking G* av G

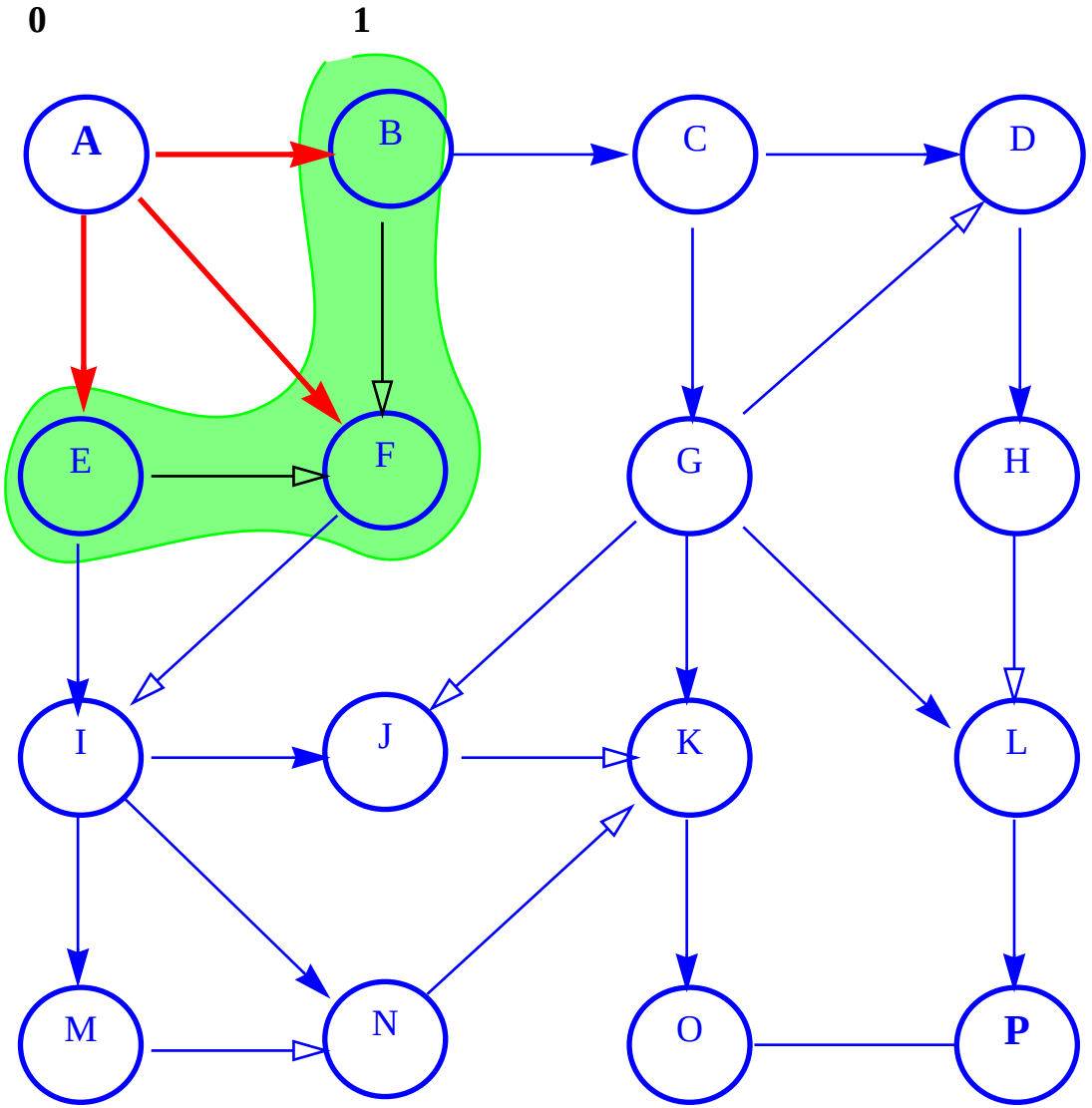
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



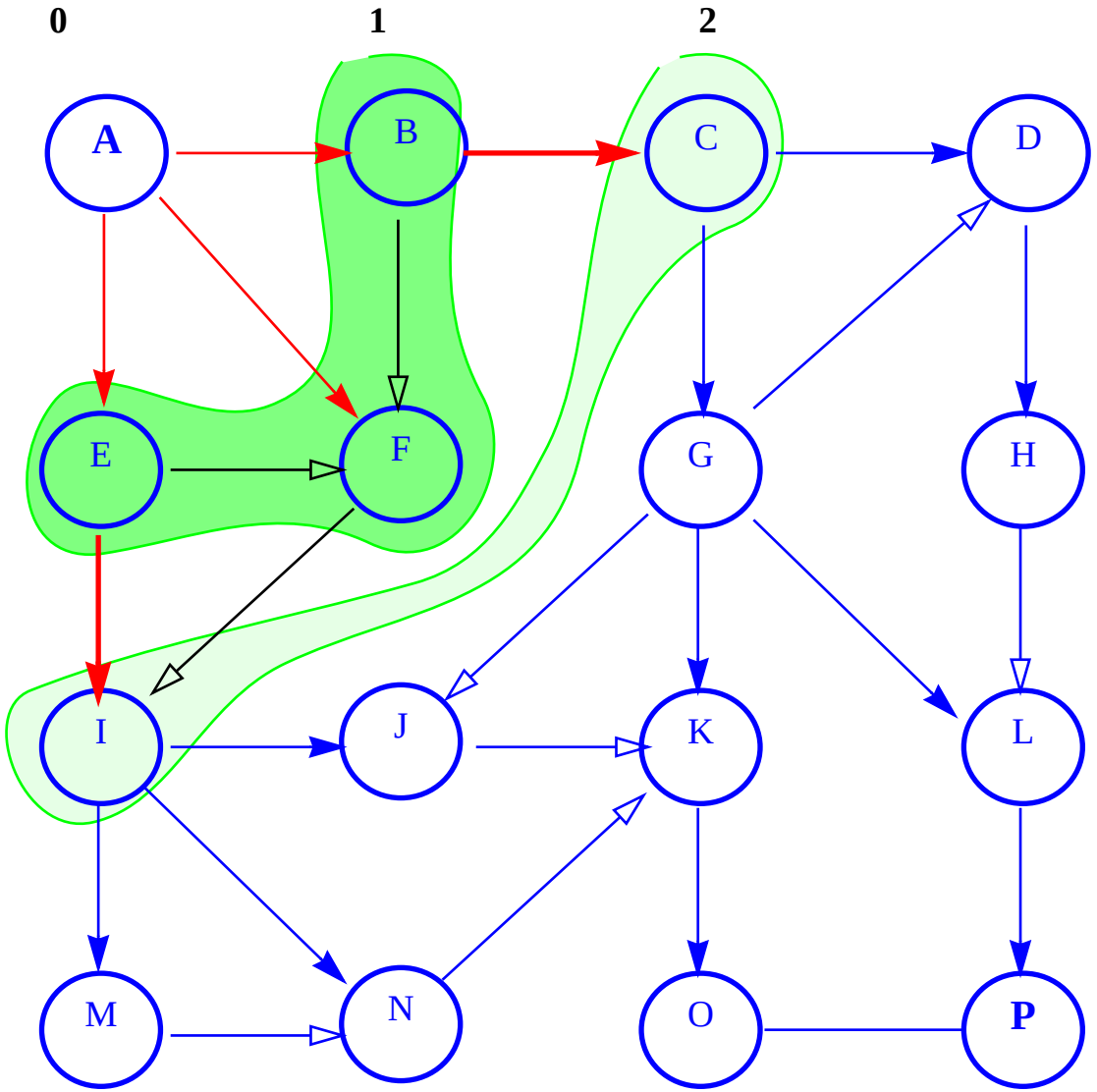
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



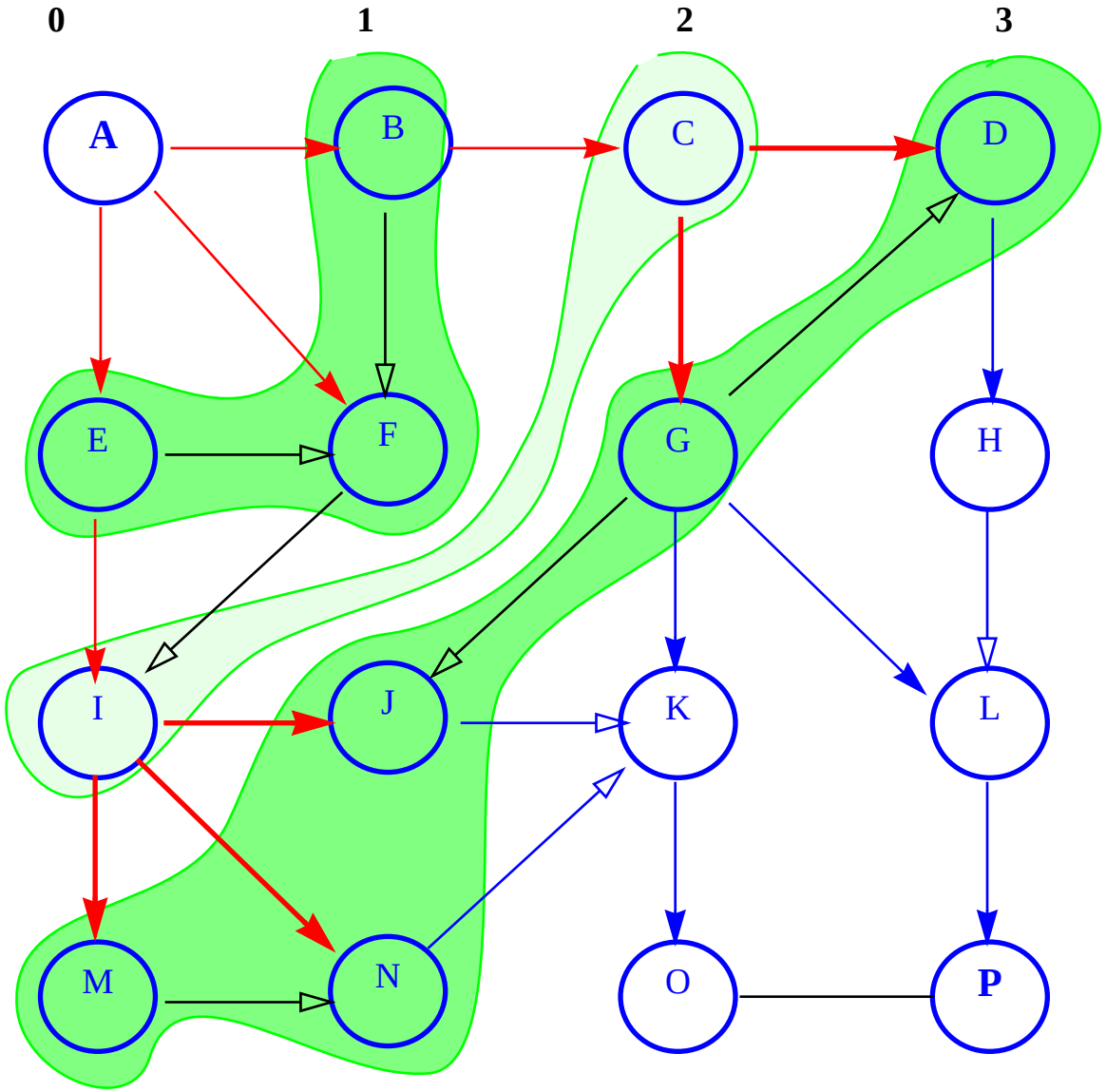
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



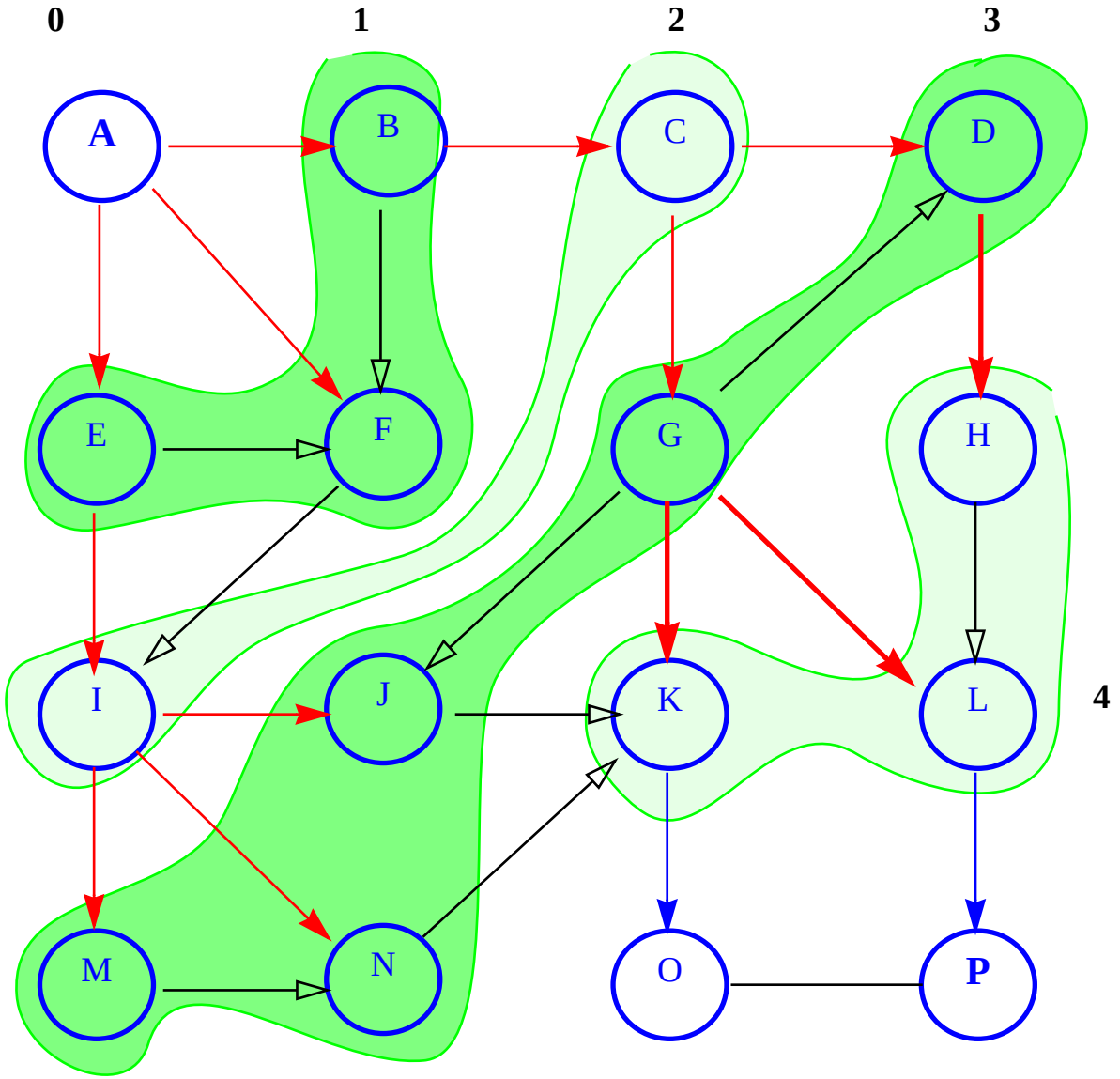
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



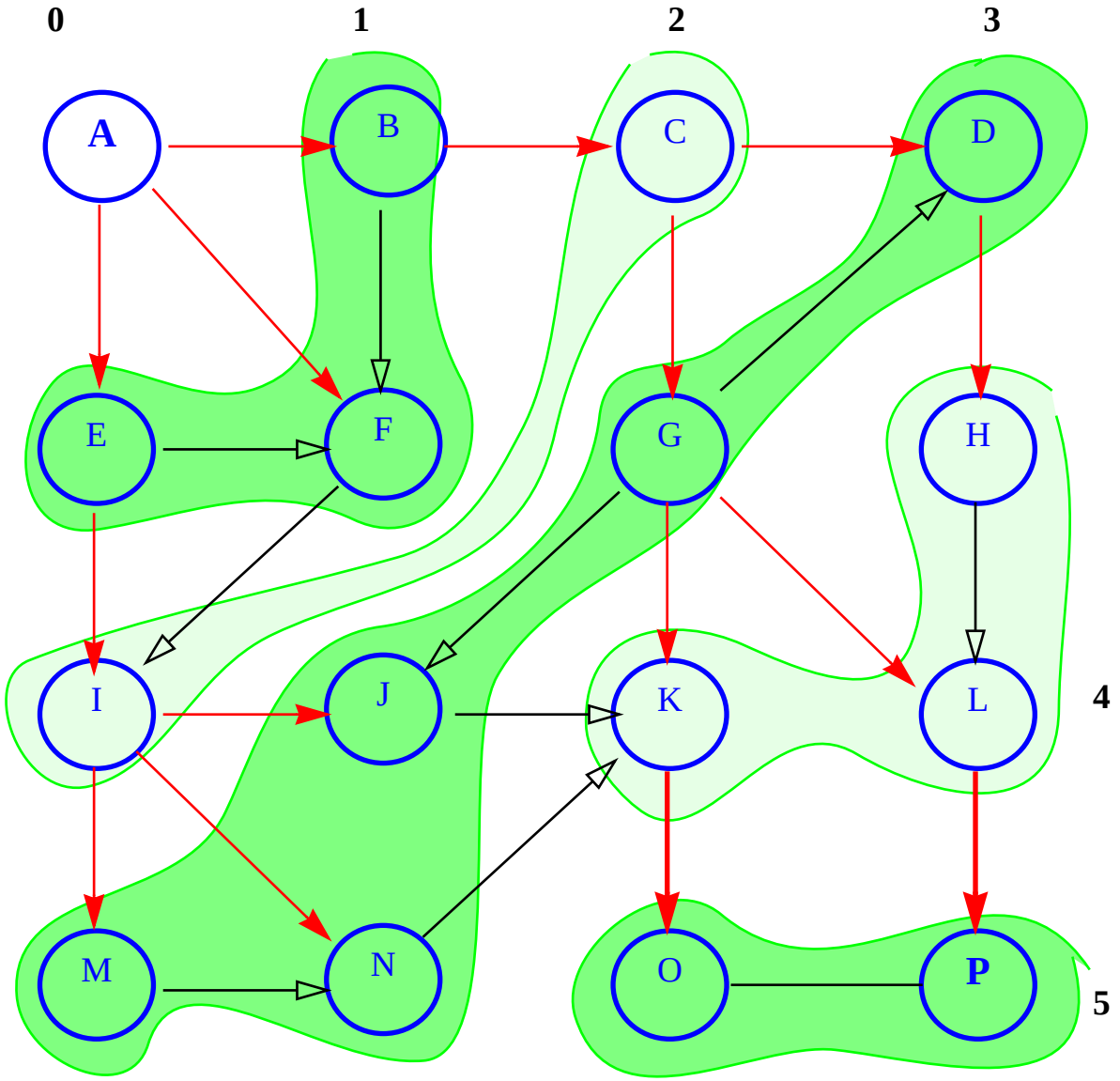
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



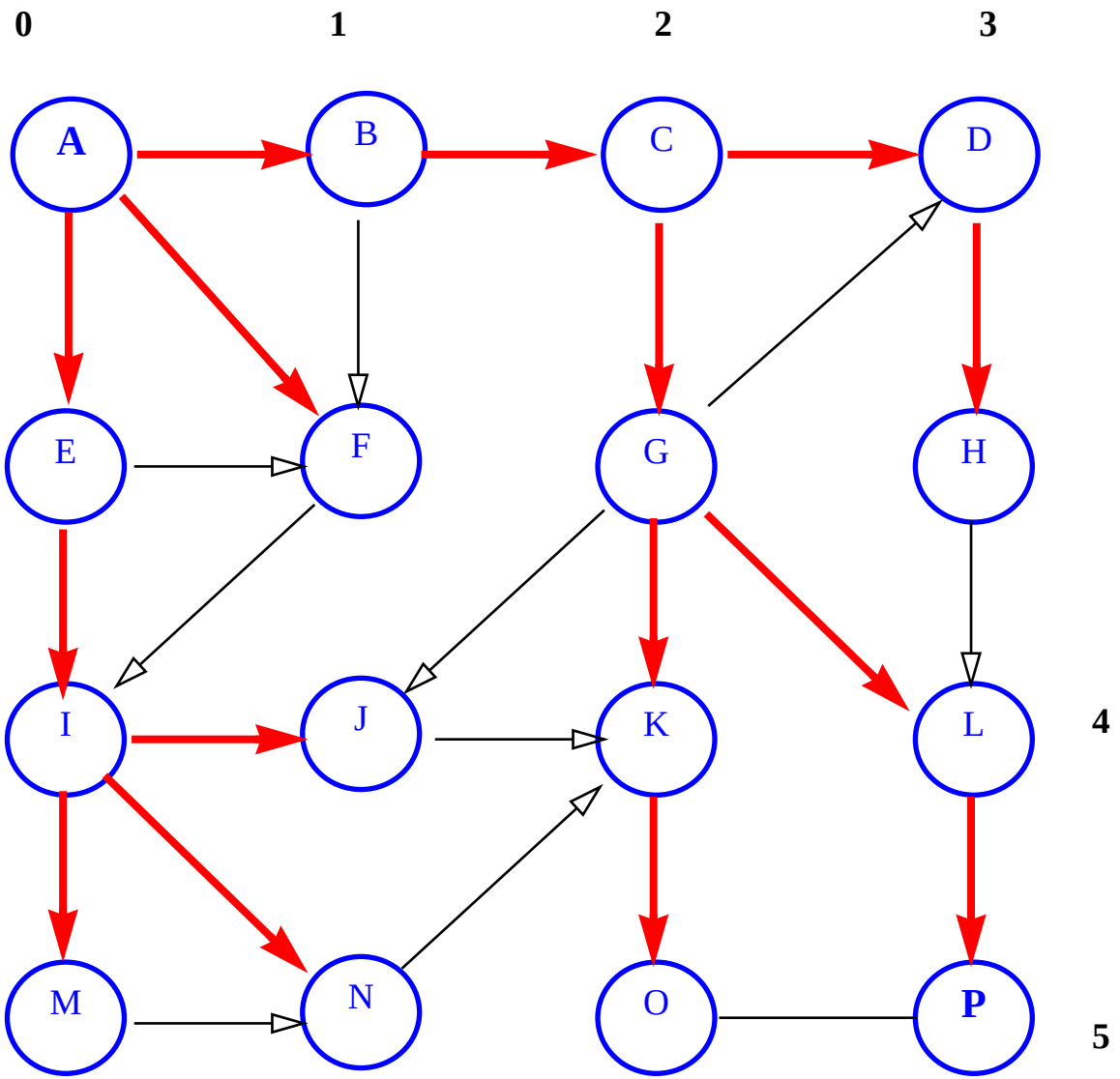
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



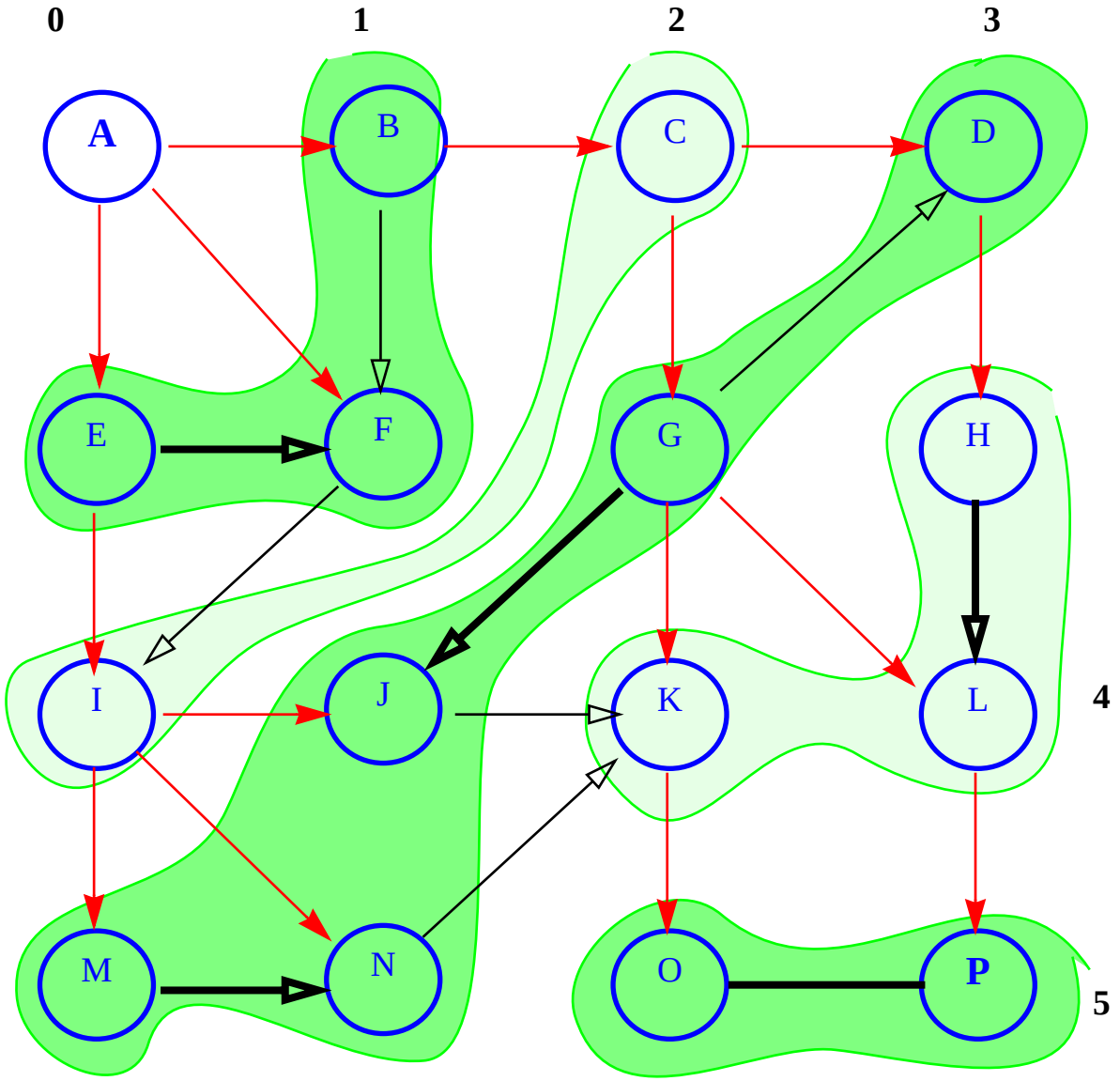
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



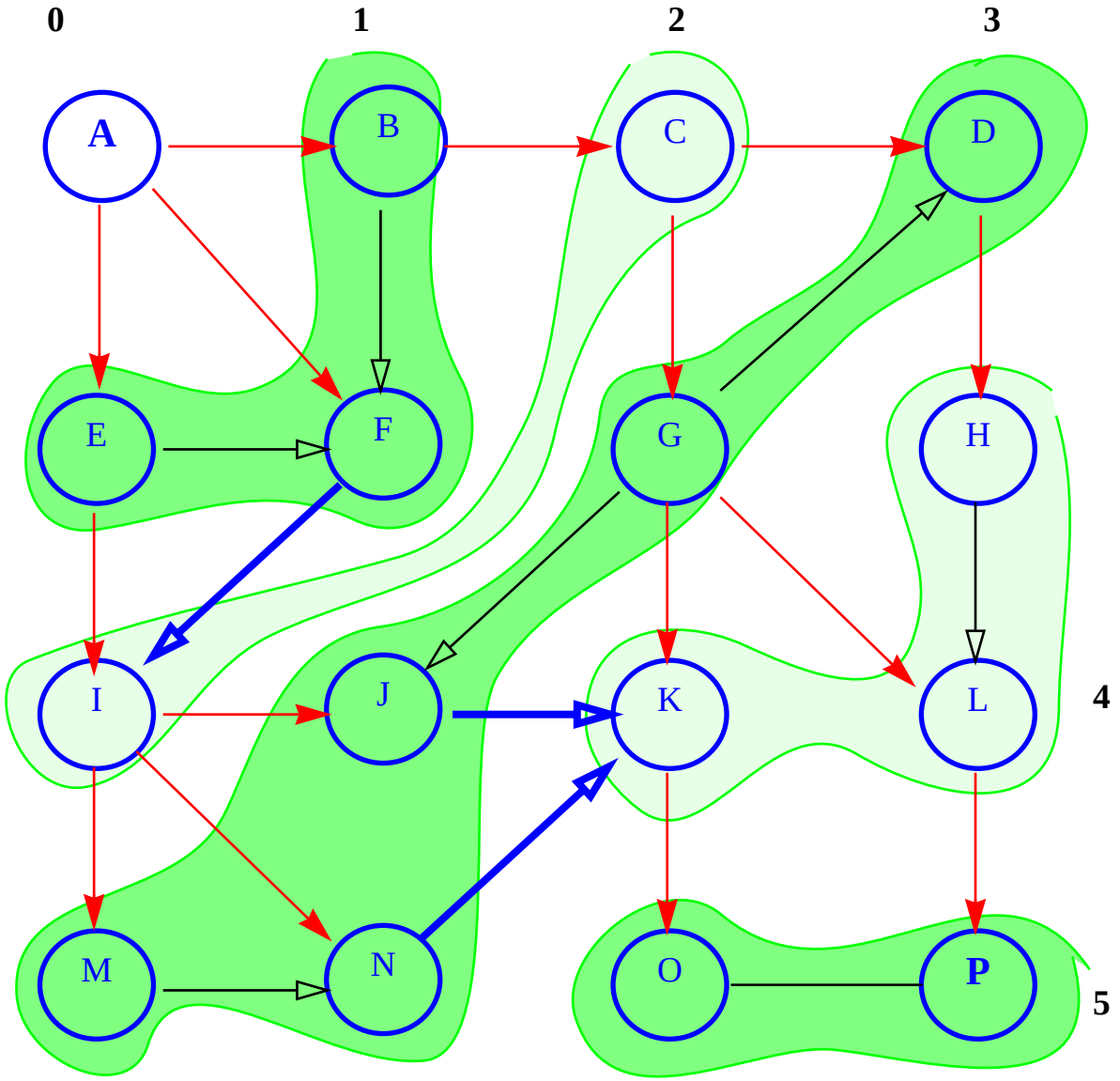
III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*

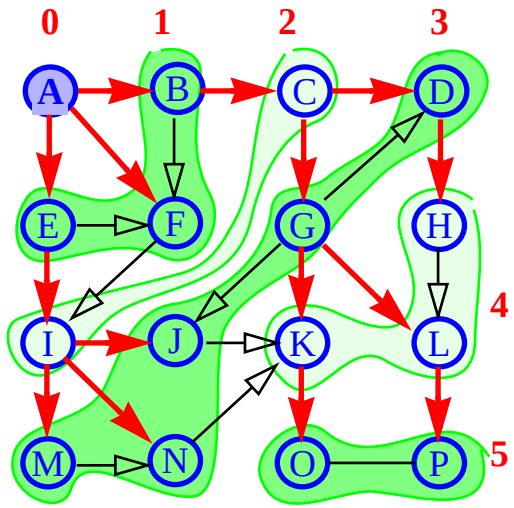


III.b) BFS traversering av grafer

Hva er lengden av korteste vei fra A *til* P *?*



III.b) BFS graf traversering



12.14. BFS traversering av en ikke-rettet graf G fra en node s:

a) besøker alle noder i en sammenhengende komponent til s

b) røde kanter danner et utspennende tre, så kalt BFS tre, for G

c) korteste stien fra **s** til hver node på nivå ***i*** har ***i*** kanter og enhver annen sti har minst ***i*** kanter

d) er ikke kant **(u,v)** med i BFS tre, så er nivå forskjellen mellom **u** og **v** høyst ***1***

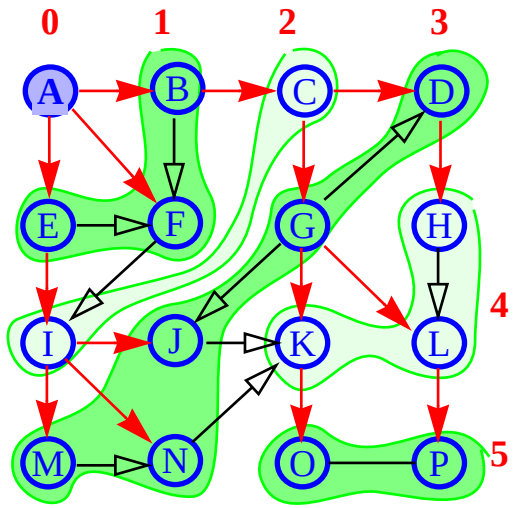
III.b) BFS graf traversering

Tree DFS

```
/*DFS(Tree T,Position s)
  *   Stack S = new StackIM()
  *   S.push(s)
  *   while (!S.isEmpty())
  *      p= S.pop()
  *      for each p's child c
  *           S.push(c)
  */
```

Tree BFS

```
/*BFS(Tree T,Position s)
  *   Queue S = new QueueIM()
  *   S.enqueue(s)
  *   while (!S.isEmpty())
  *      p= S.dequeue()
  *      for each p's child c
  *           S.enqueue(c)
  */
```



12.14. BFS traversering av en ikke-rettet graf G fra en node s:

a) besøker alle noder i en sammenhengende komponent til s

b) røde kanter danner et utspennende tre, så kalt BFS tre, for G

c) korteste stien fra **s** til hver node på nivå ***i*** har ***i*** kanter og enhver annen sti har minst ***i*** kanter

d) er ikke kant **(u,v)** med i BFS tre, så er nivå forskjellen mellom **u** og **v** høyst ***1***

III.b) BFS graf traversering

Tree DFS

```
/*DFS(Tree T,Position s)
  *   Stack S = new StackIM()
  *   S.push(s)
  *   while (!S.isEmpty())
  *      p= S.pop()
  *      for each p's child c
  *           S.push(c)
  */
```

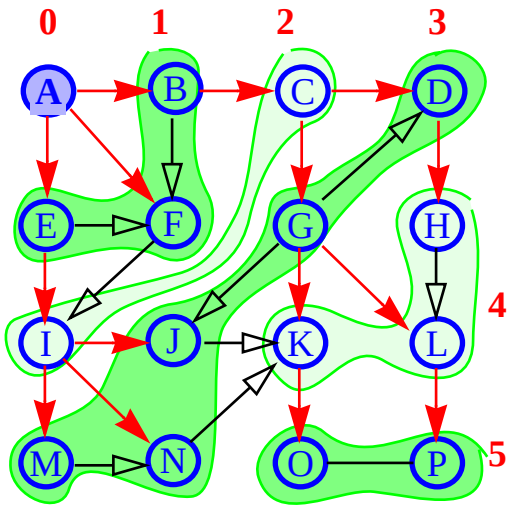
Tree BFS

```
/*BFS(Tree T,Position s)
  *   Queue S = new QueueIM()
  *   S.enqueue(s)
  *   while (!S.isEmpty())
  *      p= S.dequeue()
  *      for each p's child c
  *           S.enqueue(c)
  */
```

Graph BFS

```
// initielt er alle noder umerket

BFS(Graph G, Position s)
    Queue S = new QueueIM()
    merk(v, 0) – merker med nivå
    S.enqueue(s)
    while (!S.isEmpty())
      k= S.dequeue()
      for each kant e=(k,m)
        if (! merket(m))
            merk(m, k.merke+1)
            merk(e, rød)
            S.enqueue(m)
```



12.14. BFS traversering av en ikke-rettet graf G fra en node s:

a) besøker alle noder i en sammenhengende komponent til s

b) røde kanter danner et utspennende tre, så kalt BFS tre, for G

c) korteste stien fra **s** til hver node på nivå ***i*** har ***i*** kanter og enhver annen sti har minst ***i*** kanter

d) er ikke kant **(u,v)** med i BFS tre, så er nivå forskjellen mellom **u** og **v** høyst ***1***

III.b) BFS graf traversering

Tree DFS

```
/*DFS(Tree T,Position s)
  *   Stack S = new StackIM()
  *   S.push(s)
  *   while (!S.isEmpty())
  *      p= S.pop()
  *      for each p's child c
  *           S.push(c)
  */
```

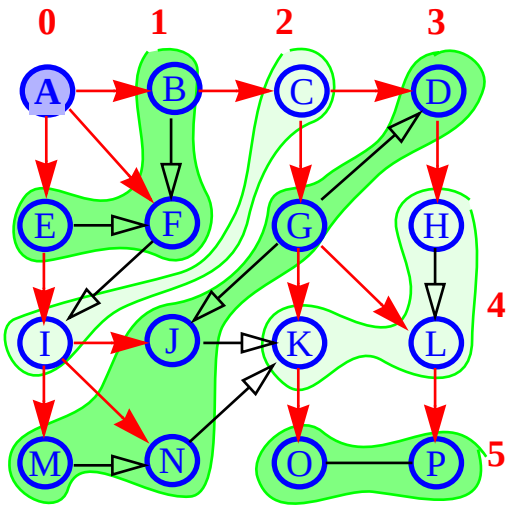
Tree BFS

```
/*BFS(Tree T,Position s)
  *   Queue S = new QueueIM()
  *   S.enqueue(s)
  *   while (!S.isEmpty())
  *      p= S.dequeue()
  *      for each p's child c
  *           S.enqueue(c)
  */
```

Graph BFS

```
// initielt er alle noder umerket

BFS(Graph G, Position s)
    Queue S = new QueueIM()
    merk(v, 0) – merker med nivå
    S.enqueue(s)
    while (!S.isEmpty())
      k = S.dequeue()
      for each kant e = (k , m)
        if (! merket(m))
            merk(m, k.merke+1)
            merk(e, rød)
            S.enqueue(m)
```



12.14. BFS traversering av en ikke-rettet graf G fra en node s:

a) besøker alle noder i en sammenhengende komponent til s

b) røde kanter danner et utspennende tre, så kalt BFS tre, for G

c) korteste stien fra **s** til hver node på nivå ***i*** har ***i*** kanter og enhver annen sti har minst ***i*** kanter

d) er ikke kant **(u,v)** med i BFS tre, så er nivå forskjellen mellom **u** og **v** høyst ***1***

III.b) BFS graf traversering

Tree DFS

```
/*DFS(Tree T,Position s)
  *   Stack S = new StackIM()
  *   S.push(s)
  *   while (!S.isEmpty())
  *      p= S.pop()
  *      for each p's child c
  *           S.push(c)
  */
```

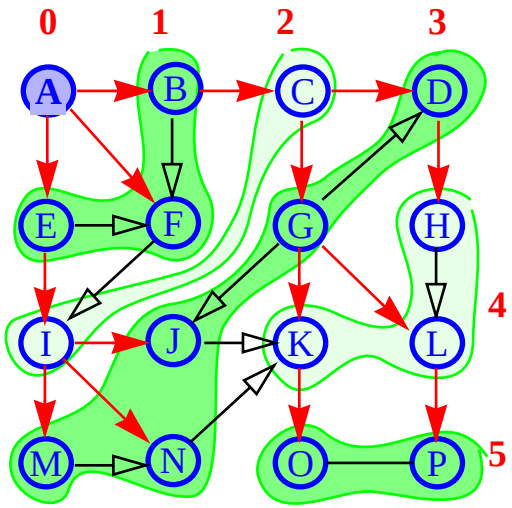
Tree BFS

```
/*BFS(Tree T,Position s)
  *   Queue S = new QueueIM()
  *   S.enqueue(s)
  *   while (!S.isEmpty())
  *      p= S.dequeue()
  *      for each p's child c
  *           S.enqueue(c)
  */
```

Graph BFS

```
// initielt er alle noder umerket

BFS(Graph G, Position s)
    Queue S = new QueueIM()
    merk(v, 0) – merker med nivå
    S.enqueue(s)
    while (!S.isEmpty())
      k = S.dequeue()
      for each kant e = (k , m)
        if (! merket(m))
            merk(m, k.merke+1)
            merk(e, rød)
            S.enqueue(m)          O(n+k)
```



12.14. BFS traversering av en ikke-rettet graf G fra en node s:

a) besøker alle noder i en sammenhengende komponent til s

b) røde kanter danner et utspennende tre, så kalt BFS tre, for G

c) korteste stien fra **s** til hver node på nivå ***i*** har ***i*** kanter og enhver annen sti har minst ***i*** kanter

d) er ikke kant **(u,v)** med i BFS tre, så er nivå forskjellen mellom **u** og **v** høyst ***1***

III.b) BFS graf traversering

Tree DFS

```
/*DFS(Tree T,Position s)
  *   Stack S = new StackIM()
  *   S.push(s)
  *   while (!S.isEmpty())
  *      p= S.pop()
  *      for each p's child c
  *           S.push(c)
  */
```

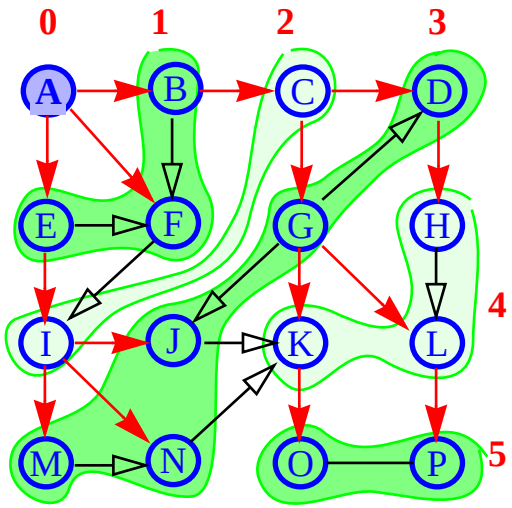
Tree BFS

```
/*BFS(Tree T,Position s)
  *   Queue S = new QueueIM()
  *   S.enqueue(s)
  *   while (!S.isEmpty())
  *      p= S.dequeue()
  *      for each p's child c
  *           S.enqueue(c)
  */
```

Graph BFS

```
// initielt er alle noder umerket

BFS(Graph G, Position s)
    Queue S = new QueueIM()
    merk(v, 0) – merker med nivå
    S.enqueue(s)
    while (!S.isEmpty())
      k= S.dequeue()
      for each kant e=(k,m)
        if (! merket(m))
            merk(m, k.merke+1)
            merk(e, rød)
            S.enqueue(m)
```



12.14. BFS traversering av en ikke-rettet graf G fra en node s:

a) besøker alle noder i en sammenhengende komponent til s

b) røde kanter danner et utspennende tre, så kalt BFS tre, for G

c) korteste stien fra **s** til hver node på nivå ***i*** har ***i*** kanter og enhver annen sti har minst ***i*** kanter

d) er ikke kant **(u,v)** med i BFS tre, så er nivå forskjellen mellom **u** og **v** høyst ***1***

BFS gir opphav til $O(n+k)$ algoritmer for å

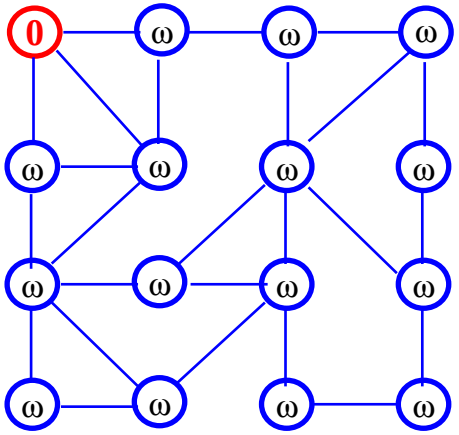
- sjekke om G er sammenhengende
- finne sammenhengende komponenter i G
- finne utspennende tre for G
- beregne minimalt antall kanter mellom to noder (korteste sti)

III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for (i=1; i<n; i++)             // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])   D[m] = D[k] + 1
```

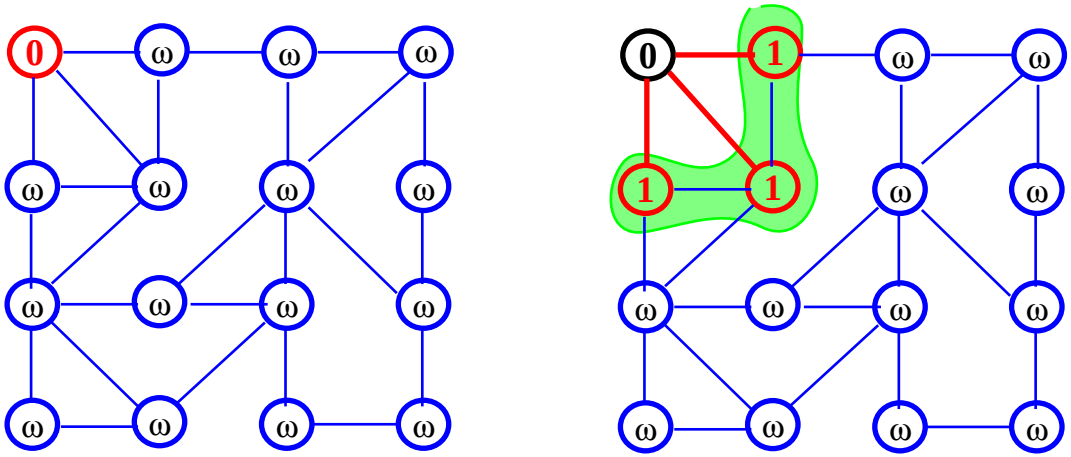
III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for (i=1; i<n; i++)             // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])   D[m] = D[k] + 1
```



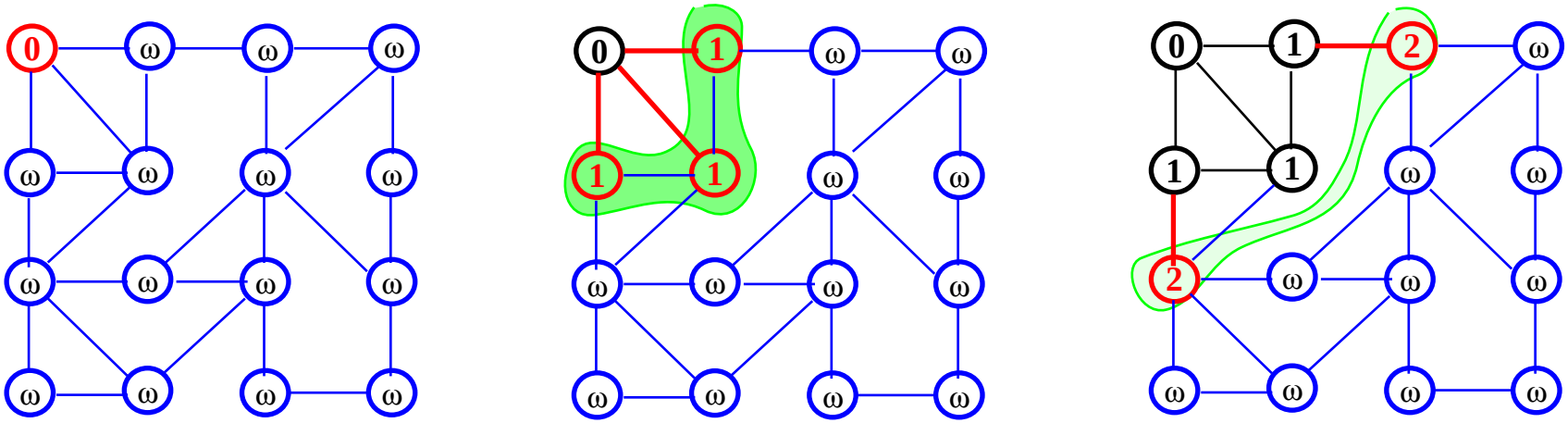
III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for (i=1; i<n; i++)             // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])  D[m] = D[k] + 1
```



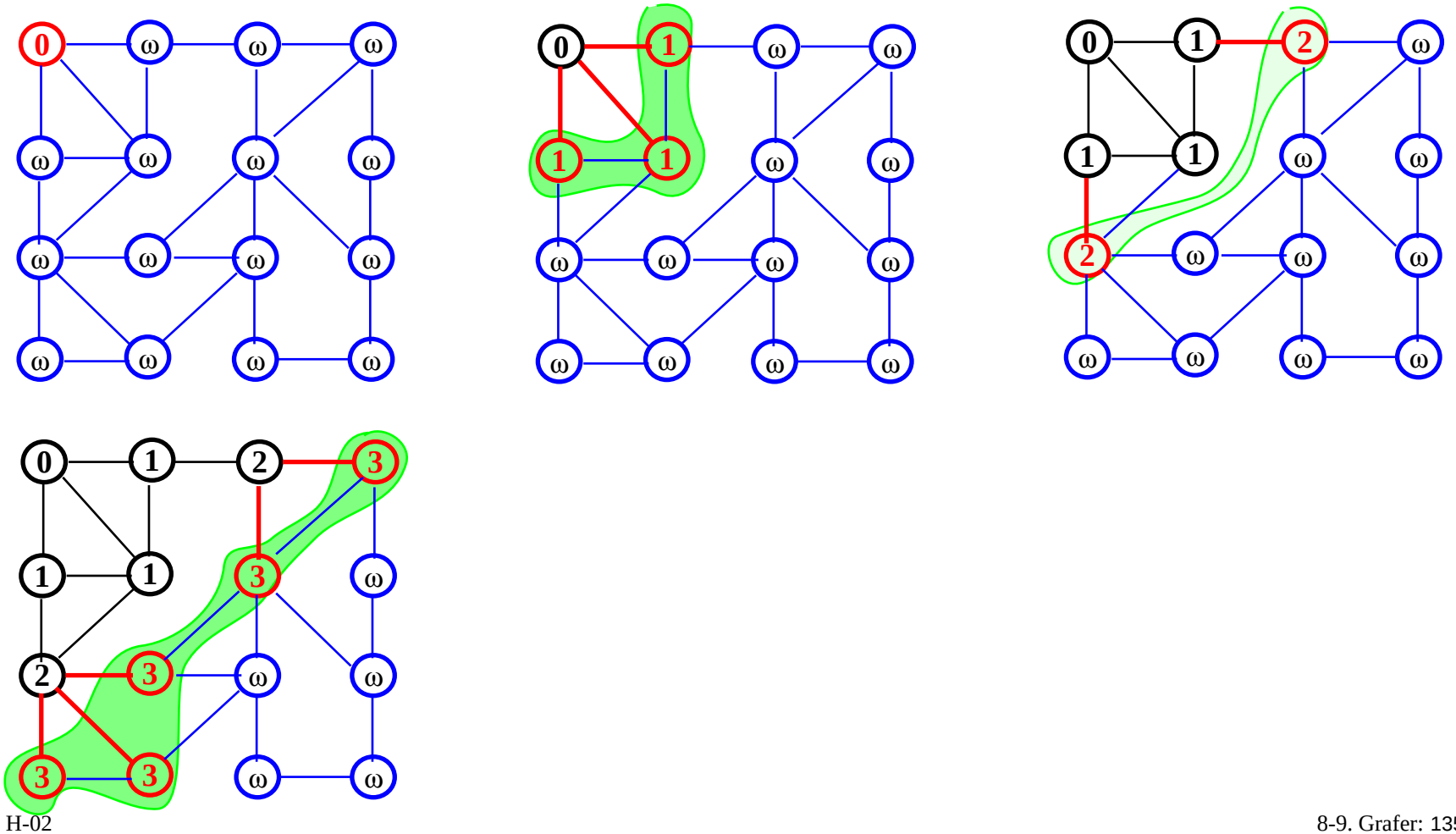
III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for (i=1; i<n; i++)             // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])  D[m] = D[k] + 1
```



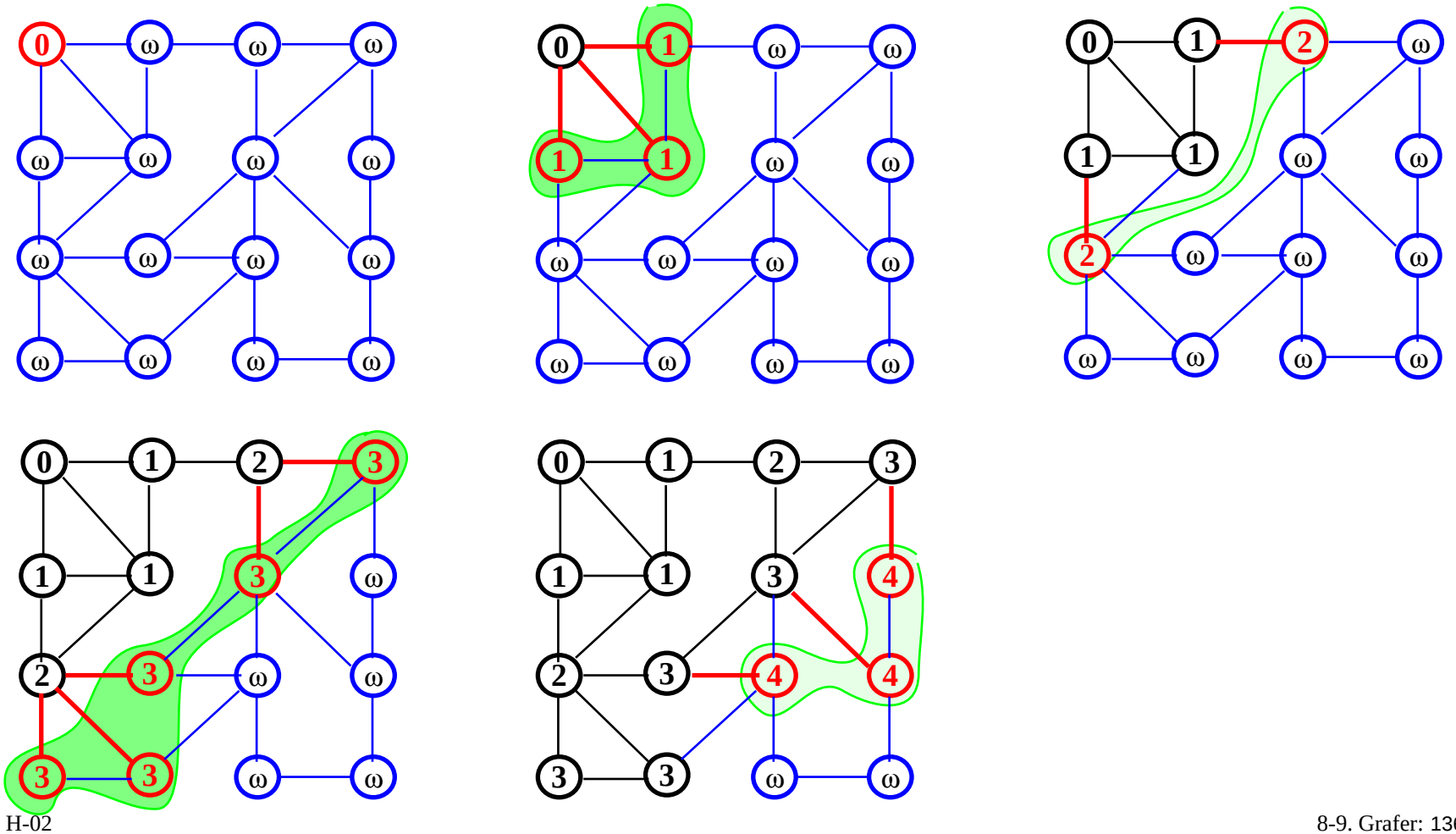
III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for (i=1; i<n; i++)             // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])  D[m] = D[k] + 1
```



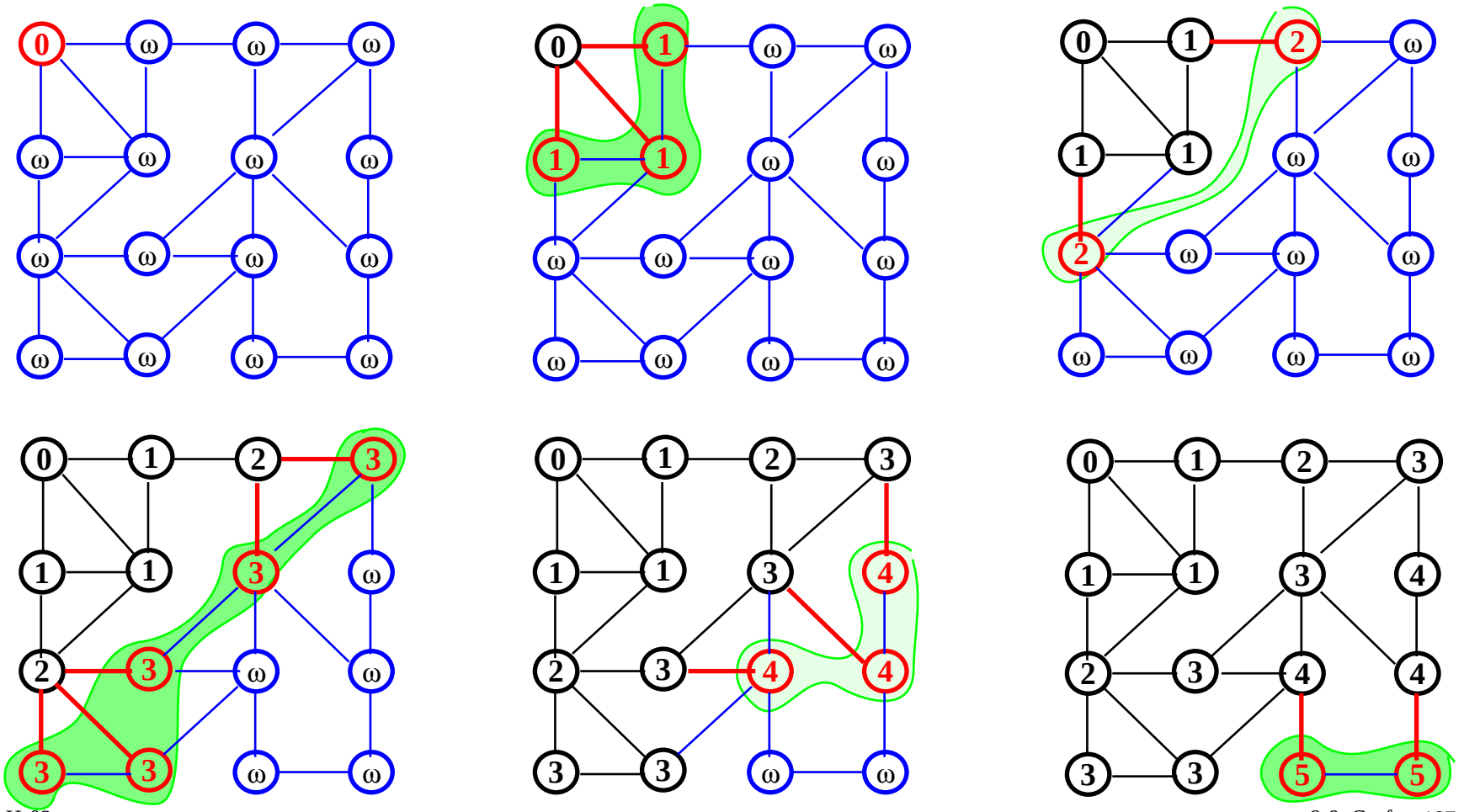
III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for (i=1; i<n; i++)             // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])  D[m] = D[k] + 1
```



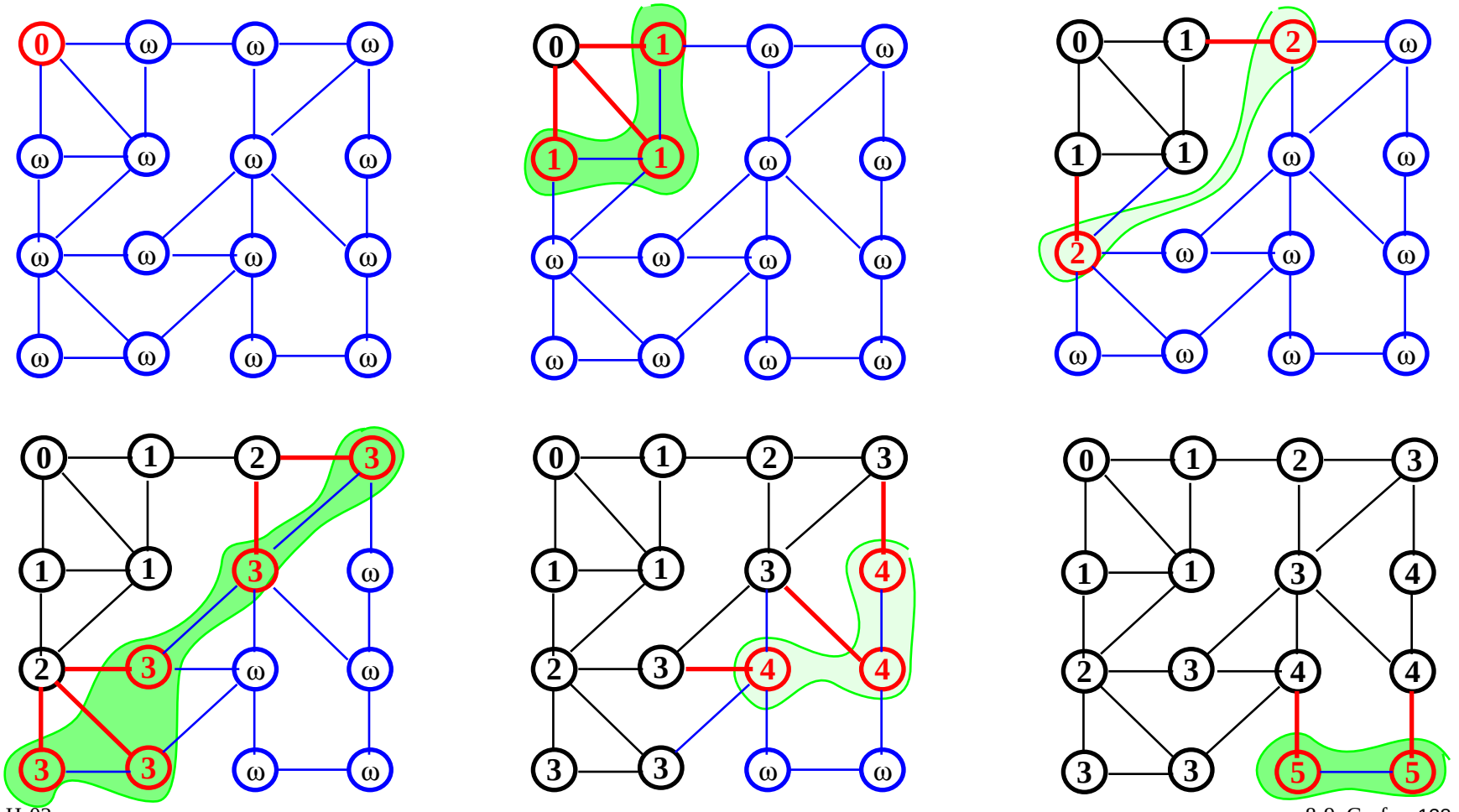
III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for (i=1; i<n; i++)             // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])  D[m] = D[k] + 1
```



III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω       // ω er et maksimalt tall (her ω > n)
D[s] = 0;                       // s er startnoden
for ( i=1;  i < n;  i++ )       // n er antall noder i grafen G
   for each kant (k,m)          // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])   D[m] = D[k] + 1
```



III.b) BFS: Ford-Bellman

```
for each node v : D[v]= ω      // ω er et maksimalt tall (her ω > n)
D[s] = 0;                      // s er startnoden
for ( i=1;  i < n;  i++ )      // n er antall noder i grafen G                    O(n*k)
   for each kant (k,m)         // for en ikke-rettet G : (k,m) ε G hviss (m,k) ε G
      if ( D[k] + 1 < D[m])   D[m] = D[k] + 1
```