Forelesning 9

Her har vi en graf med vekter på kantene, og ønsker å bare beholde akkurat de kantene vi må for å koble sammen alle nodene, med en så lav vektsum som mulig. Erke-eksempel på grådighet: Velg én og én kant, alltid den billigste lovlige.

Pensum

- [ ] Kap. 21. Data structures for disjoint sets: Innledning, 21.1 og 21.3
- [ ] Kap. 23. Minimum spanning trees

Læringsmål

[I_1] Forstå *skog*-implementasjonen av *disjunkte mengder*

[I_2] Vite hva *spenntrær* og *minimale spenntrær* er

[I_3] Forstå Generic-MST

[I_4] Forstå hvorfor *lette kanter* er *trygge kanter*

[I_5] Forstå MST-Kruskal

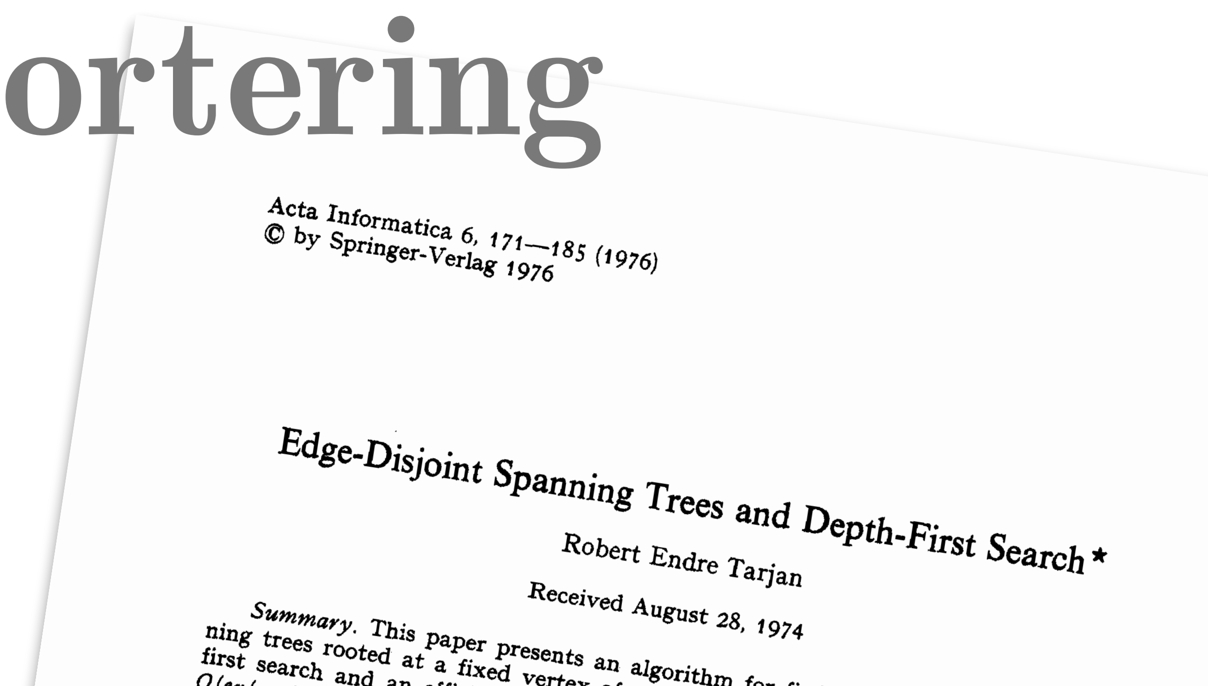
[I_6] Forstå MST-Prim

mlh@ntnu.no

Forelesningen filmes

0:4

Topologisk sortering


Acta Informatica 6, 171—185 (1976)
© by Springer-Verlag 1976

Edge-Disjoint Spanning Trees and Depth-First Search*

Robert Endre Tarjan

Received August 28, 1974

Summary. This paper presents an algorithm for

Det finnes flere algoritmer for dette – men varianten som bruker DFS ble trolig først beskrevet av Tarjan.

Kjøretid, traversering

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	
Bredde-først-søk	$\Theta(V + E)$	

Alle noder farges grå og svarte; alle kanter undersøkes

mlh@ntnu.no

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	

DFS starter fra alle noder etter tur, så topsort skal funke!

mlh@ntnu.no

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	

(Generelt går det også fint å kjøre DFS-Visit fra bare én node)

mlh@ntnu.no

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	$\Theta(V)$

Grunnen til at best-case ikke er O(1) er at vi må initialisere alle nodene. Dersom vi hadde brukt f.eks. en hashtabell til å holde denne informasjonen, og kun lagt den inn etter behov, så kunne vi ha fått O(1) – men det er altså ikke slik boka gjør det.

BFS starter ikke fra alle i vår versjon (men kunne ha gjort det)

mlh@ntnu.no

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	$\Theta(V)$

F.eks. Edmonds-Karp trenger stier fra én bestemt node

mlh@ntnu.no

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	$\Theta(V)$
______-først-søk	$\Omega(V + E)$	

Samme logikk for andre prioriteringer, men kan ha dyrere kø!

mlh@ntnu.no

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	$\Theta(V)$
______-først-søk	$\Omega(V + E)$	

Binærhaug (Prim, Dijkstra): $\Theta(V \lg V + E \lg E) =$

mlh@ntnu.no

Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V+E)$	$\Theta(V+E)$
Bredde-først-søk	$\Theta(V+E)$	$\Theta(V)$
_______-først-søk	$\Omega(V+E)$	

Binærhaug (Prim, Dijkstra): $\Theta(V \lg V + E \lg E) = \Theta(E \lg V)$

mlh@ntnu.no

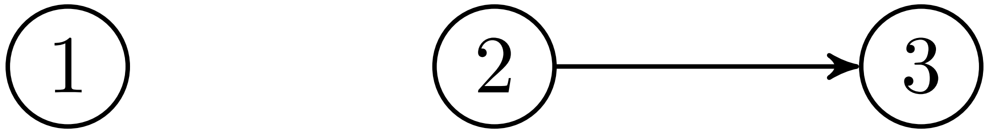
Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	$\Theta(V)$
______-først-søk	$\Omega(V + E)$	$\Theta(V)$

Bortsett fra i DFS starter vi vanligvis kun ett sted

mlh@ntnu.no

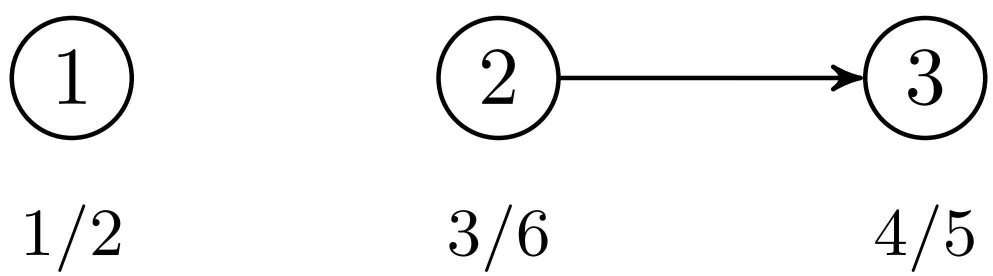
Topologisk sortering

- Gir nodene en rekkefølge
- Foreldre før barn
- Evt.: Alle kommer etter avhengigheter
- Det er egentlig det vi gjør med delproblemgrafen i dynamisk programmering
- Krever at grafen er en DAG!



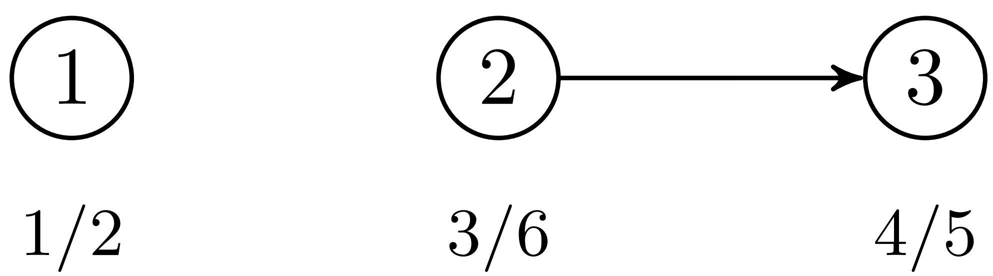
Her må 2 komme før 3; ellers fritt frem

mlh@ntnu.no



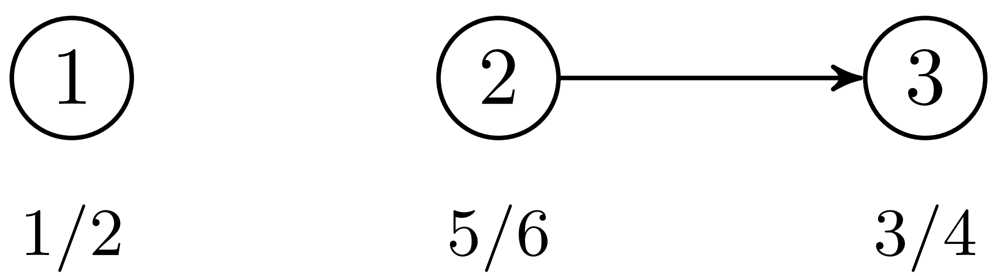
Mulige *discover-/finish-times*

mlh@ntnu.no



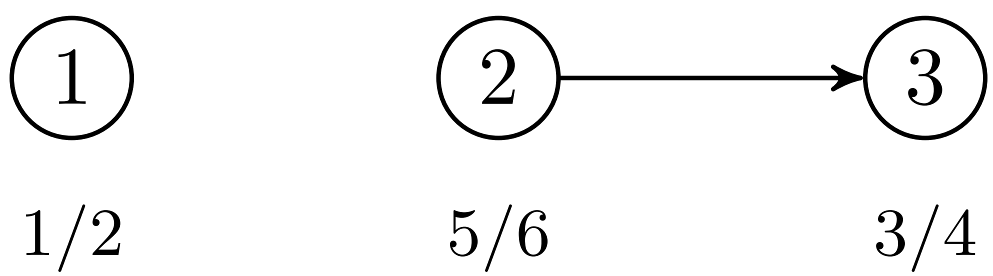
Her oppdages 2 før 3, men det er ikke garantert!

mlh@ntnu.no



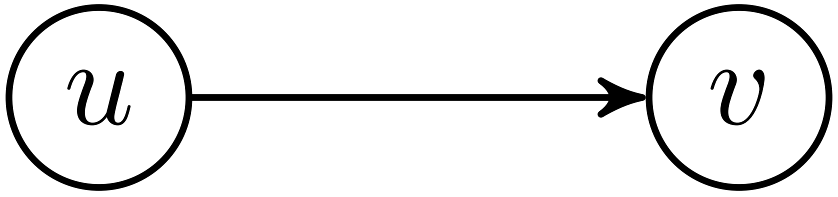
Her kalles DFS-VISIT(G, 3) før DFS-VISIT(G, 2)

mlh@ntnu.no



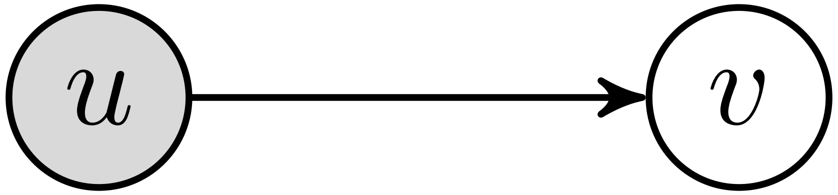
Merk at 2 fortsatt har høyere finish-time enn 3!

mlh@ntnu.no



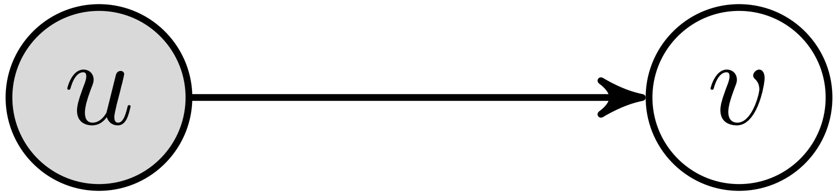
La oss si vi undersøker kanten (u, v) under DFS

mlh@ntnu.no



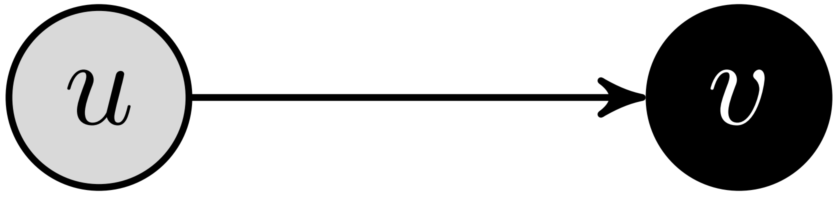
La oss si vi undersøker kanten (u, v) under DFS

mlh@ntnu.no



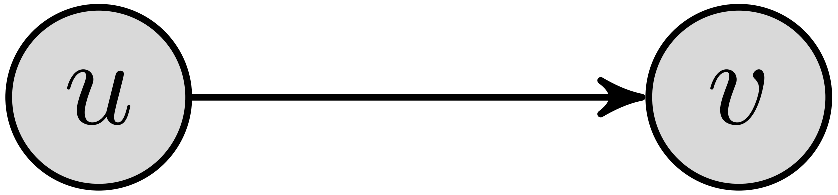
Hvis v er hvit, så utforskers den rekursivt: $u.f > v.f$

mlh@ntnu.no



Hvis v er svart, så er den alt ferdig: $u.f > v.f$

mlh@ntnu.no

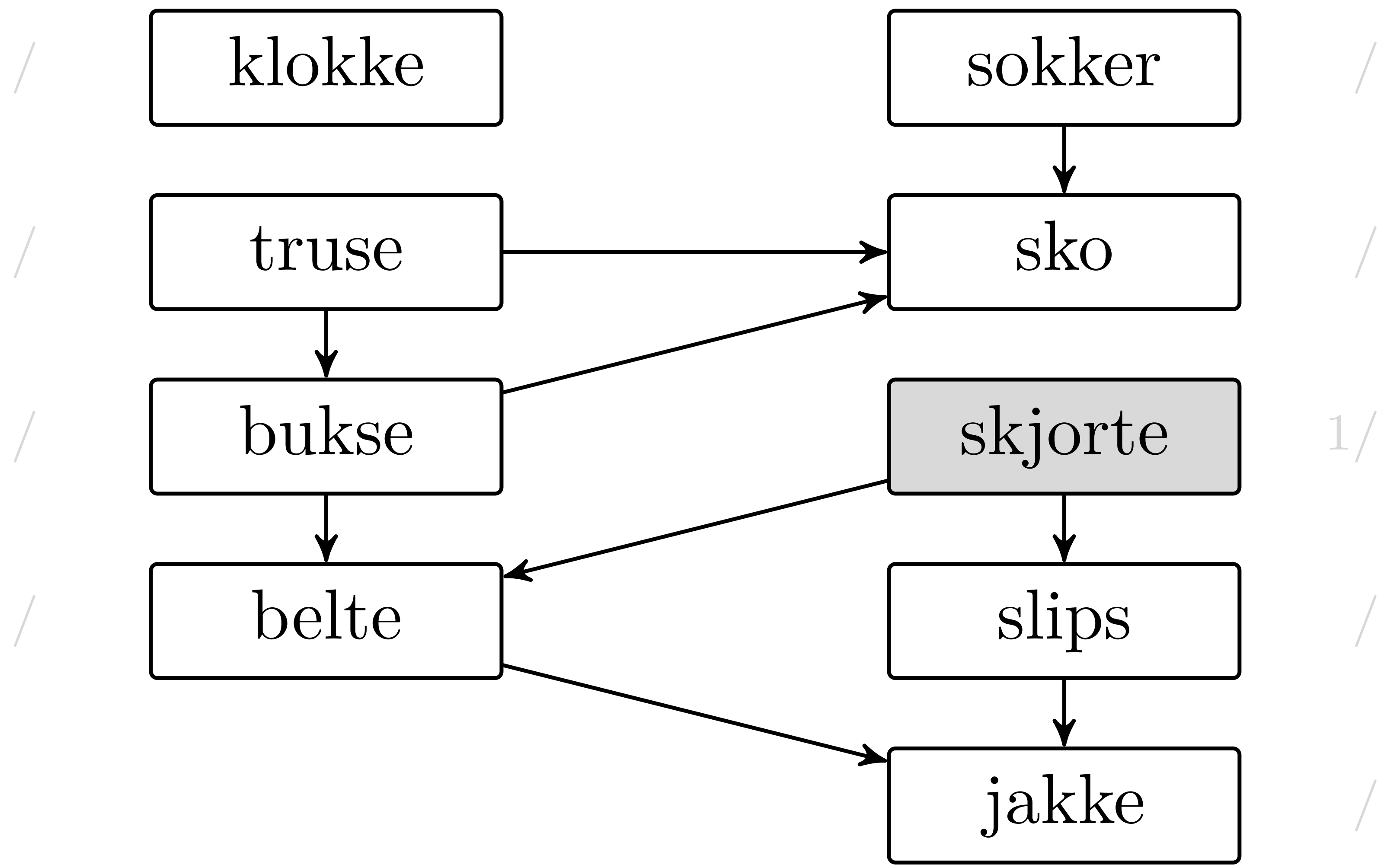


Hvis v er grå, så har vi en sykel – og det er umulig!

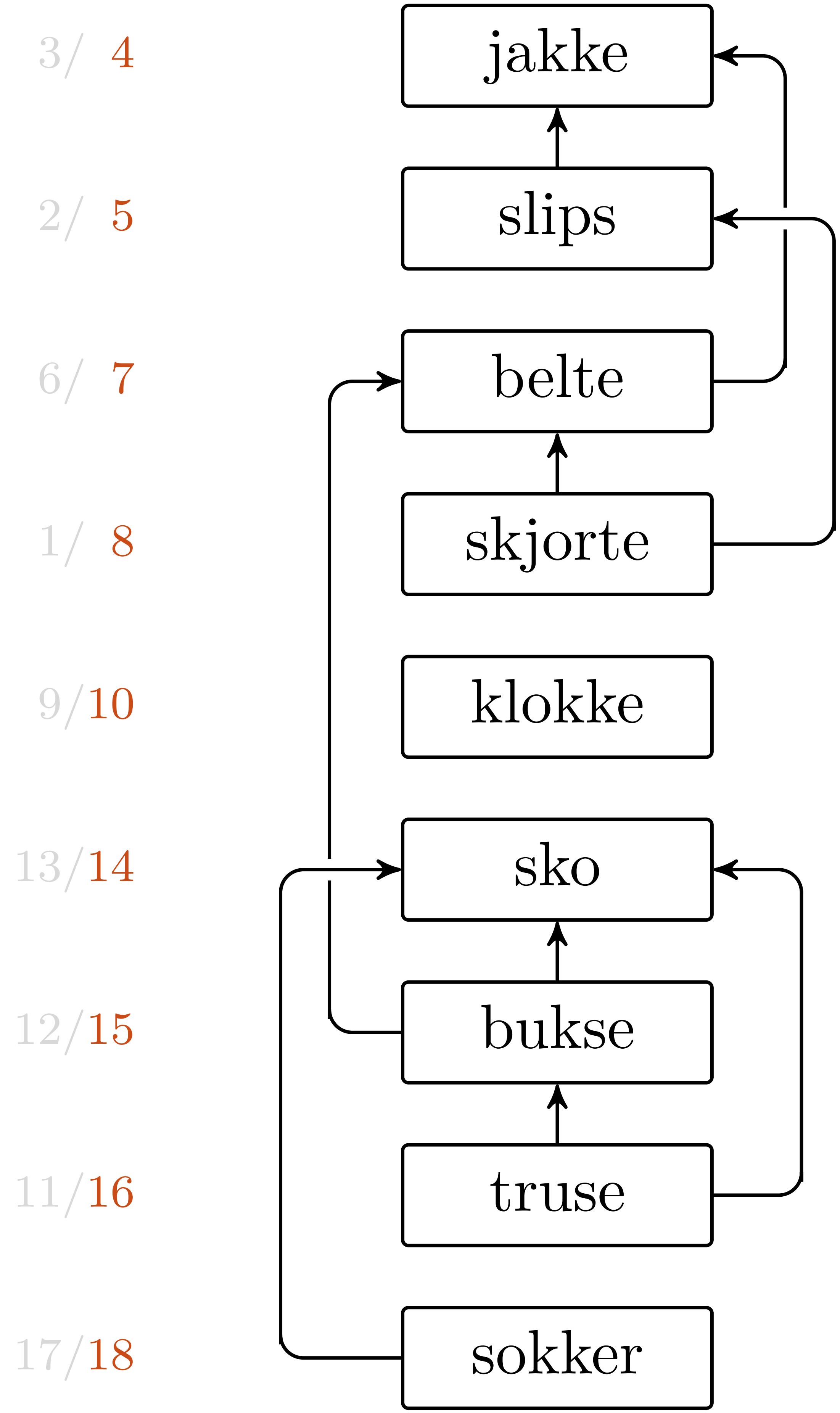
mlh@ntnu.no

Altså ...

- **Stigende discover-tid: Ikke trygt**
- **Synkende finish-tid: Trygt**
- **Dvs.: Det gir en topologisk sortering**



mlh@ntnu.no


3/ 4
jakke
2/ 5
slips
6/ 7
belte
1/ 8
skjorte
9/10
klokke
13/14
sko
12/15
bukse
11/16
truse
17/18
sokker

mlh@ntnu.no

- **I DP med memoisering: Vi utfører implisitt DFS på delproblemene**
- **Vi får automatisk en topologisk sortering: Problemer løses etter delproblemer**
- **Det samme som å sortere etter synkende finish-tid**

Tenk på selv: Hva er sammenhengen mellom pakkesystemer (som automatisk installerer programpakker og avhengigheter) og topologisk sortering ved dybde-først-søk?

Kartet er fra Prims artikkel.

Forelesning 9

Minimale spenntrær

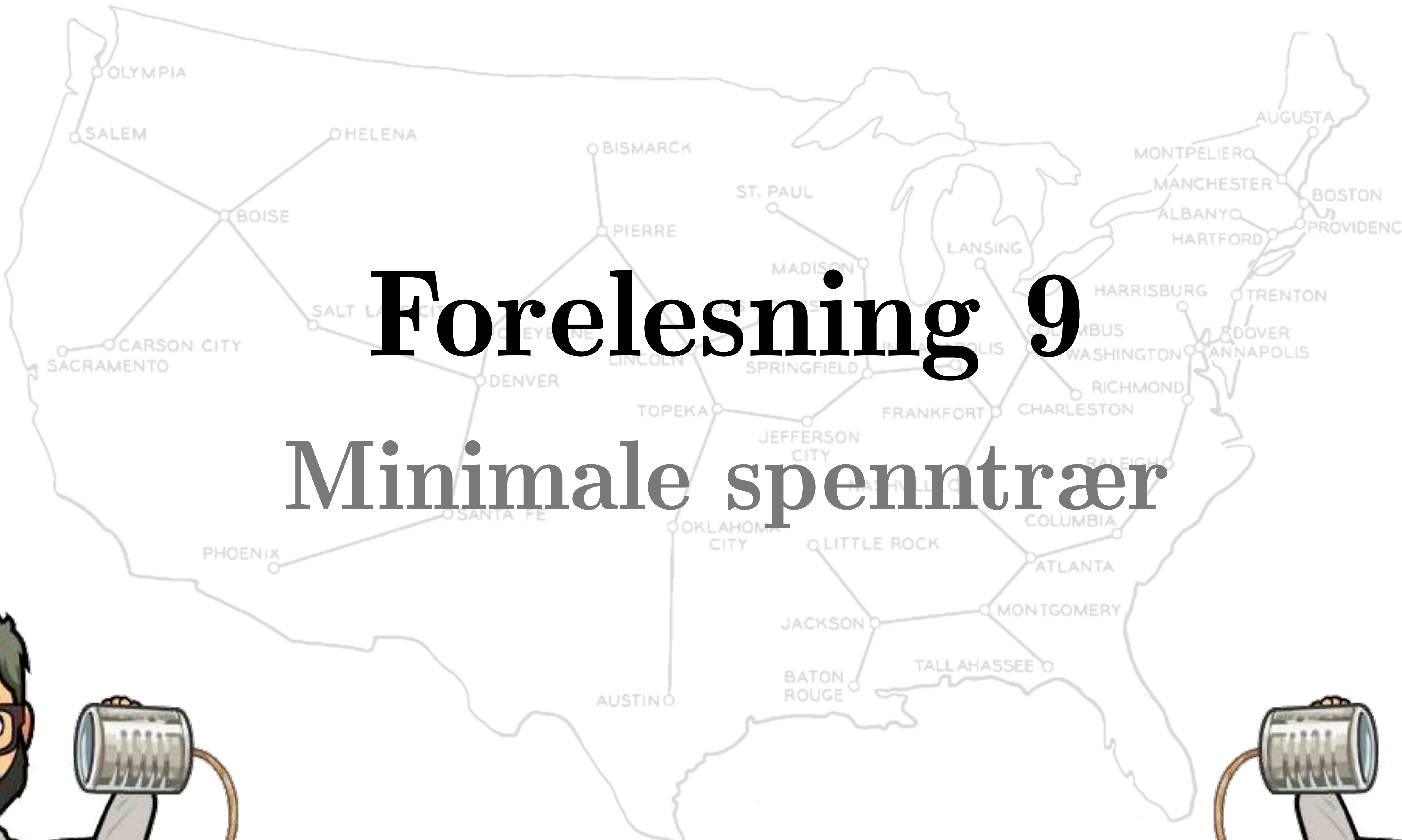


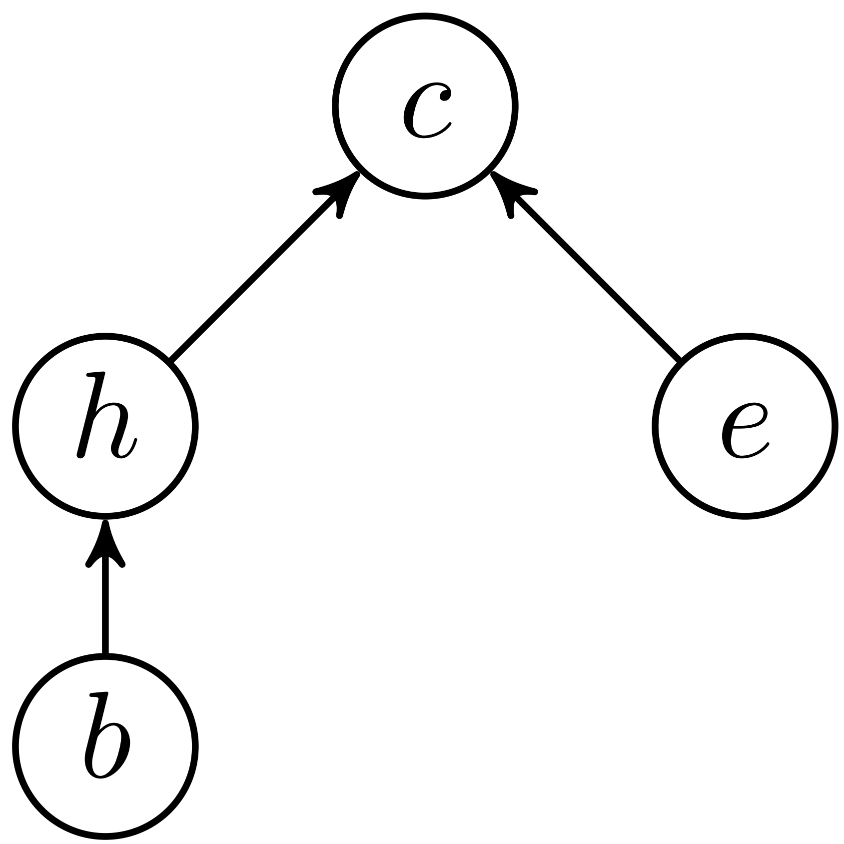
Fig. 1 — Example of a shortest connection network.

47

1. **Disjunkte mengder**
2. **Generisk MST**
3. **Kruskals algoritme**
4. **Prims algoritme**

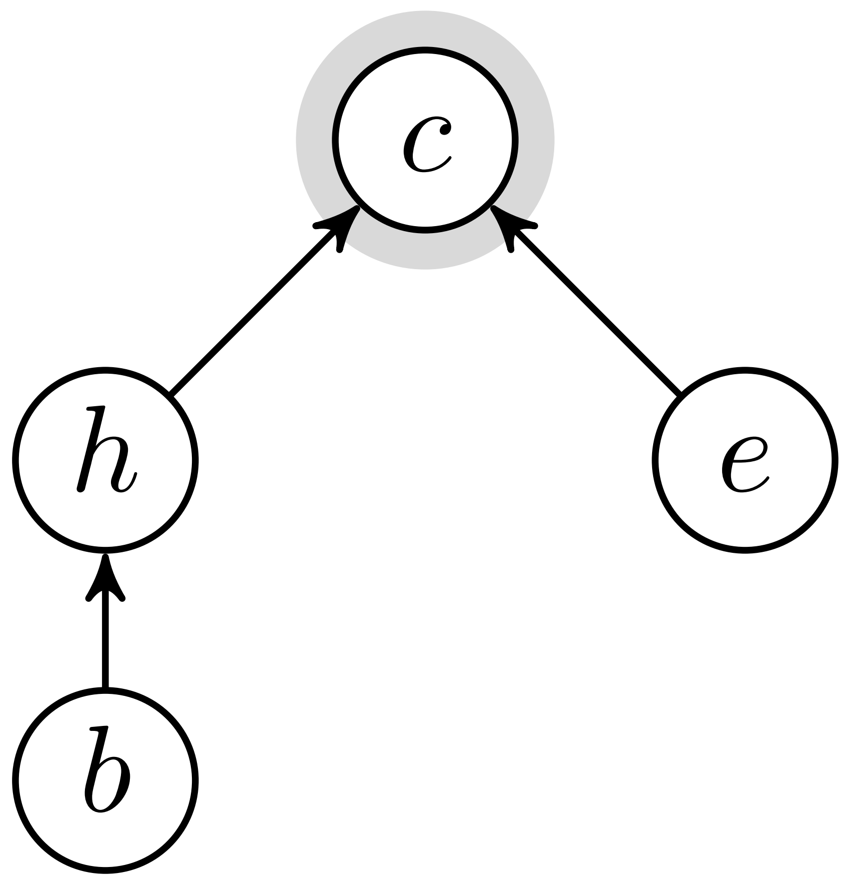
1:4

Disjunkte mengder



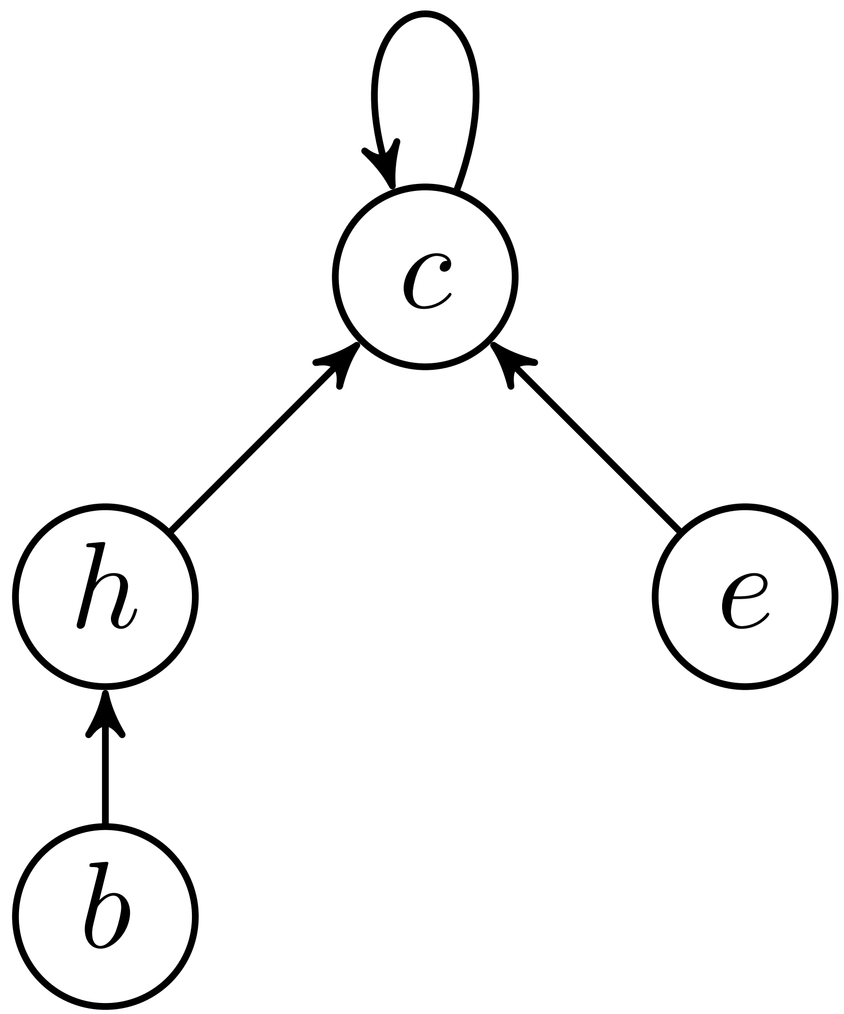
Mengder representeres som trær vha. foreldrepekere $v.p$

mlh@ntnu.no



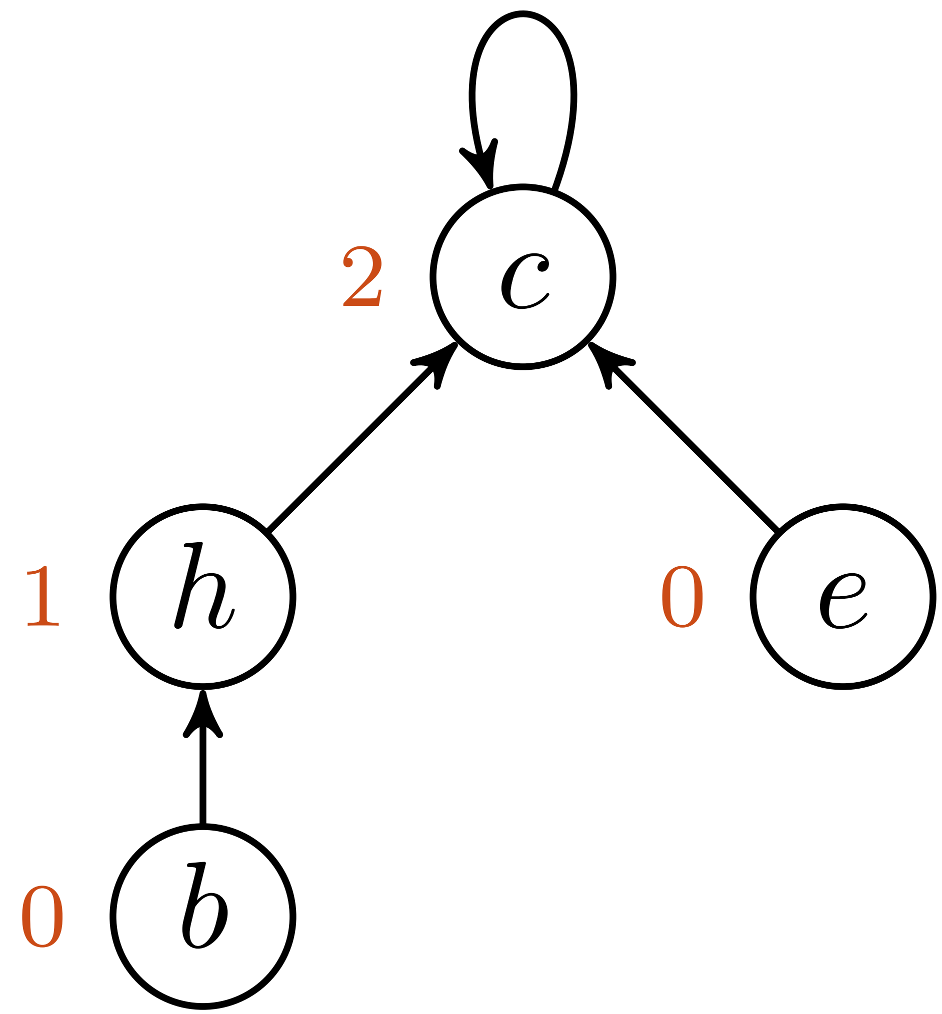
Rota representerer mengden; FIND-SET(v) gir peker til rota

mlh@ntnu.no



Self-loop: Fjerner spesialtilfelle fra FIND-SET

mlh@ntnu.no



For *union by rank*-heuristikk: Rang er øvre grense for nodehøyde

mlh@ntnu.no

MAKE-SET(x)

Initialisering: Noden representerer mengden $\{x\}$

mlh@ntnu.no

$\textsc{Make-Set}(x)$
1 $x.p = x$

Mengden representeres av et tre; dette er foreldrenoden til x

mlh@ntnu.no

Make-Set(x)

1 $x.p = x$

2 $x.rank = 0$

En slags «høyde»; største avstand til en løvnode

mlh@ntnu.no

$\textsc{Union}(x, y)$

Kombinér to mengder/trær

mlh@ntnu.no

$\textsc{Union}(x, y)$
1 $\textsc{Link}(\textsc{Find-Set}(x), \textsc{Find-Set}(y))$

Finn røttene/«representantene» og koble dem sammen

mlh@ntnu.no

$\textsc{Link}(x, y)$

To røtter; én henges under den andre

mlh@ntnu.no

$\textsc{Link}(x, y)$
1 **if** $x.rank > y.rank$

Utjevning: Den med høyest rang blir rot

mlh@ntnu.no

LINK(x, y)

if $x.rank > y.rank$

  $y.p = x$

Den nærmest «bunnen» blir barn av den andre

mlh@ntnu.no

$\textsc{Link}(x, y)$

```
if x.rank > y.rank
    y.p = x
else x.p = y
```


mlh@ntnu.no

$\textsc{Link}(x, y)$

```
if x.rank > y.rank
        y.p = x
else x.p = y
        if x.rank == y.rank
```

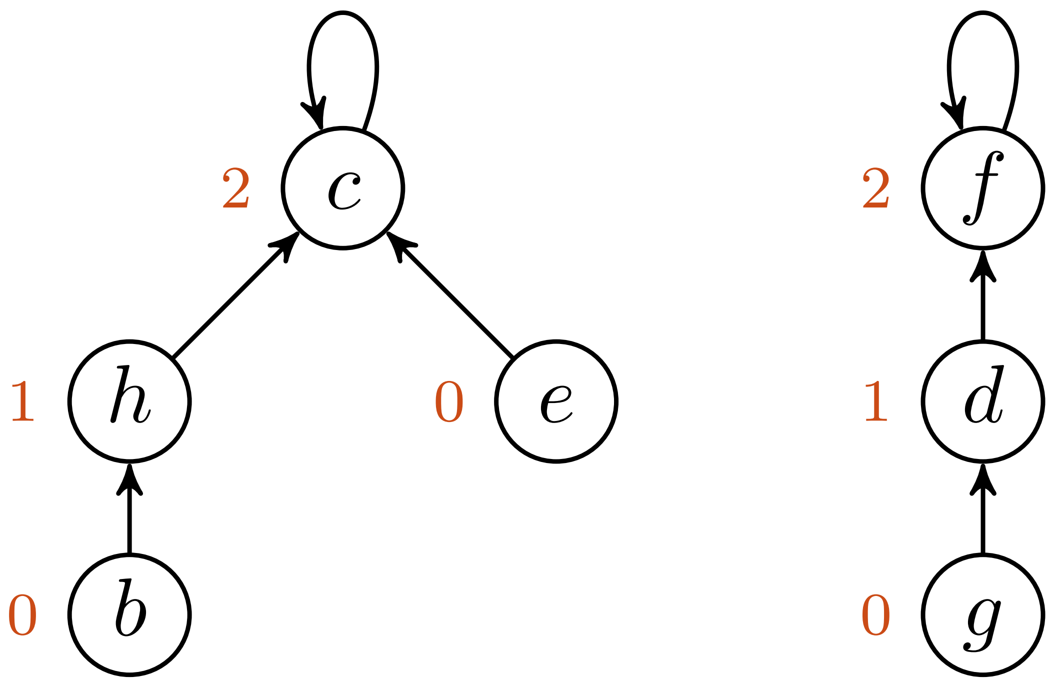
Barn strengt mindre rang? Alt i orden. Ellers . . .

mlh@ntnu.no

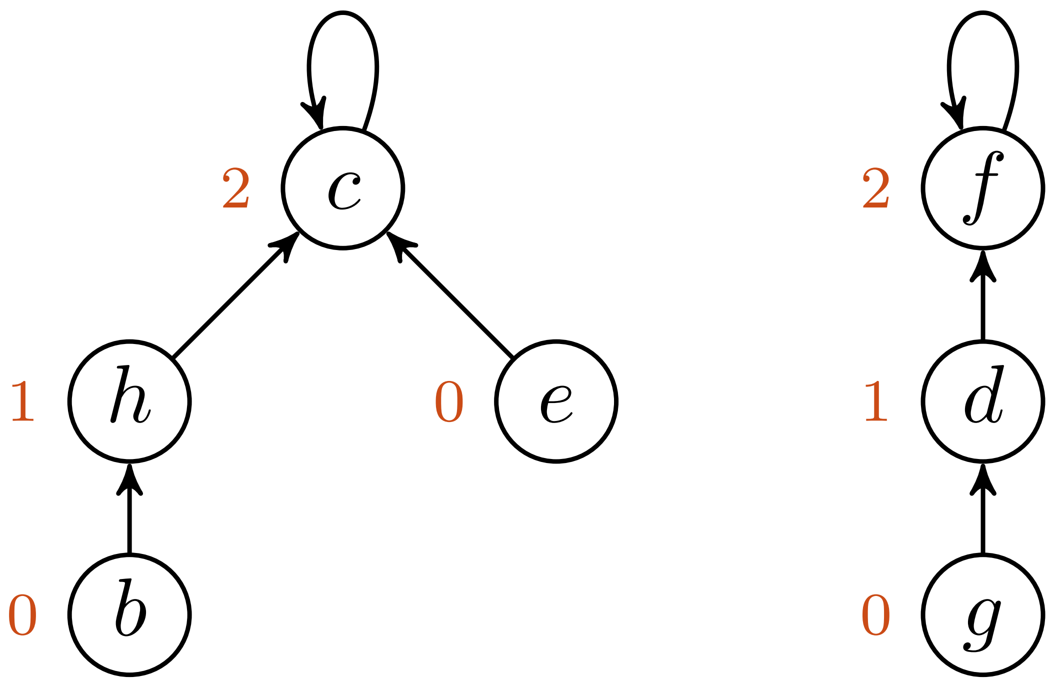
```
LINK(x, y)
if x.rank > y.rank
        y.p = x
else x.p = y
        if x.rank == y.rank
                y.rank = y.rank + 1
```

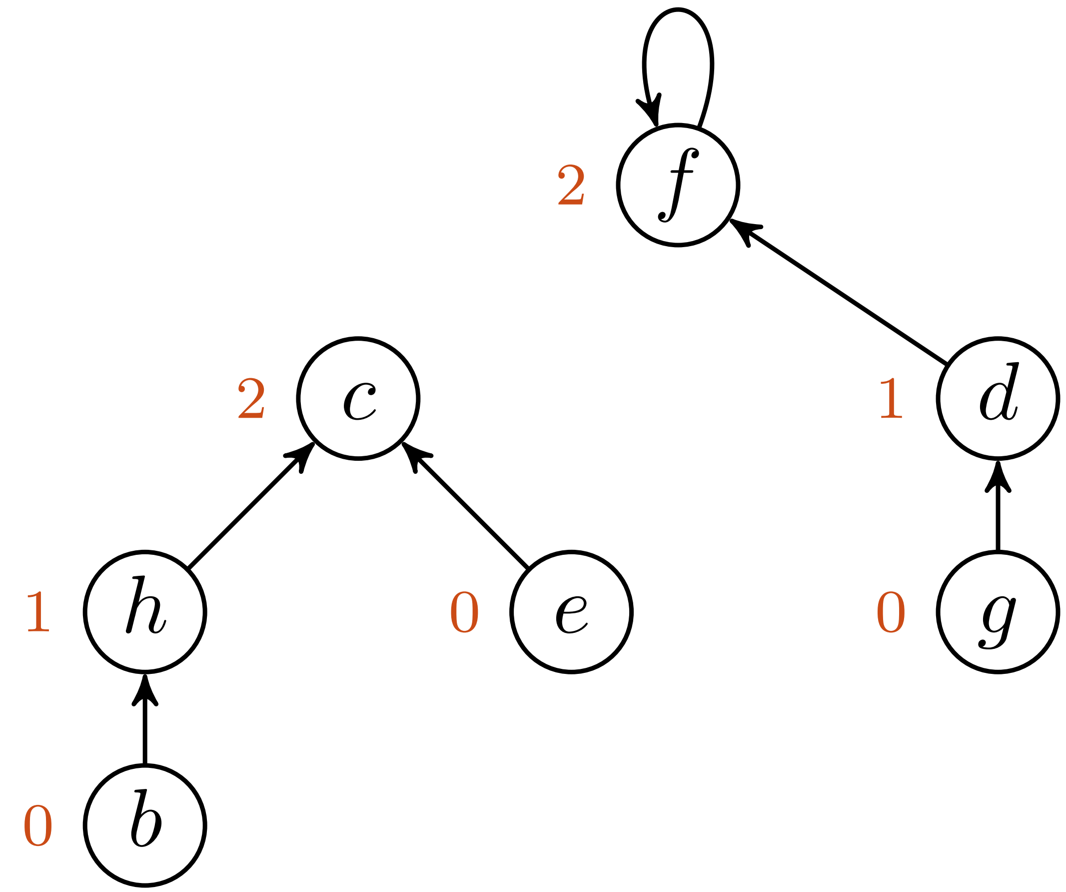
Vedlikehlold betydningen av rang: Avstand til bunnen

mlh@ntnu.no

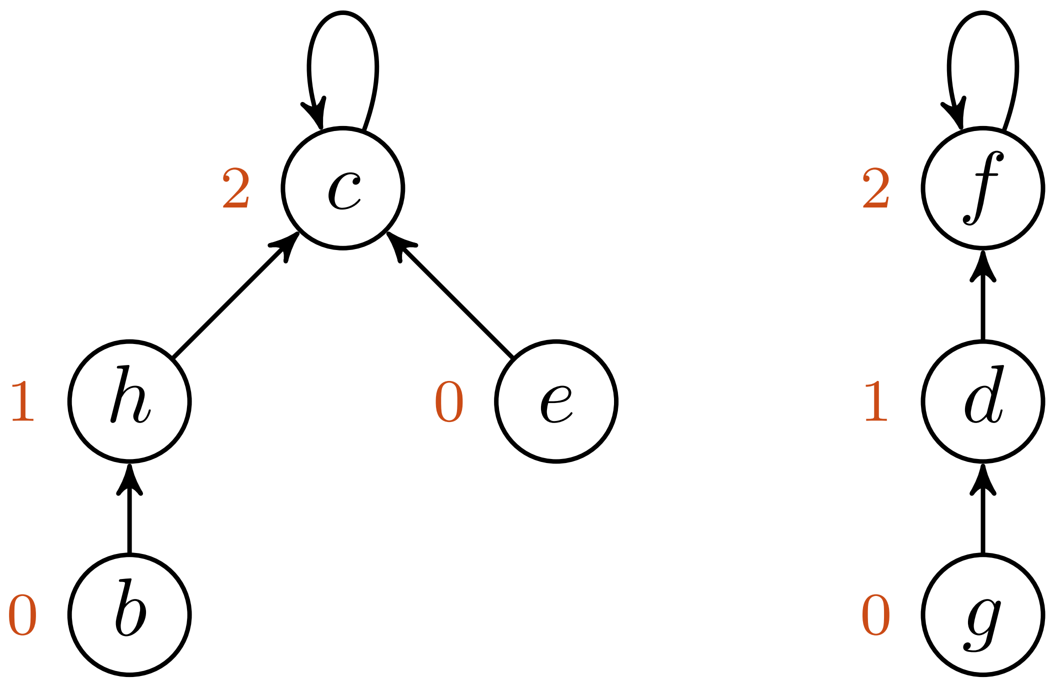


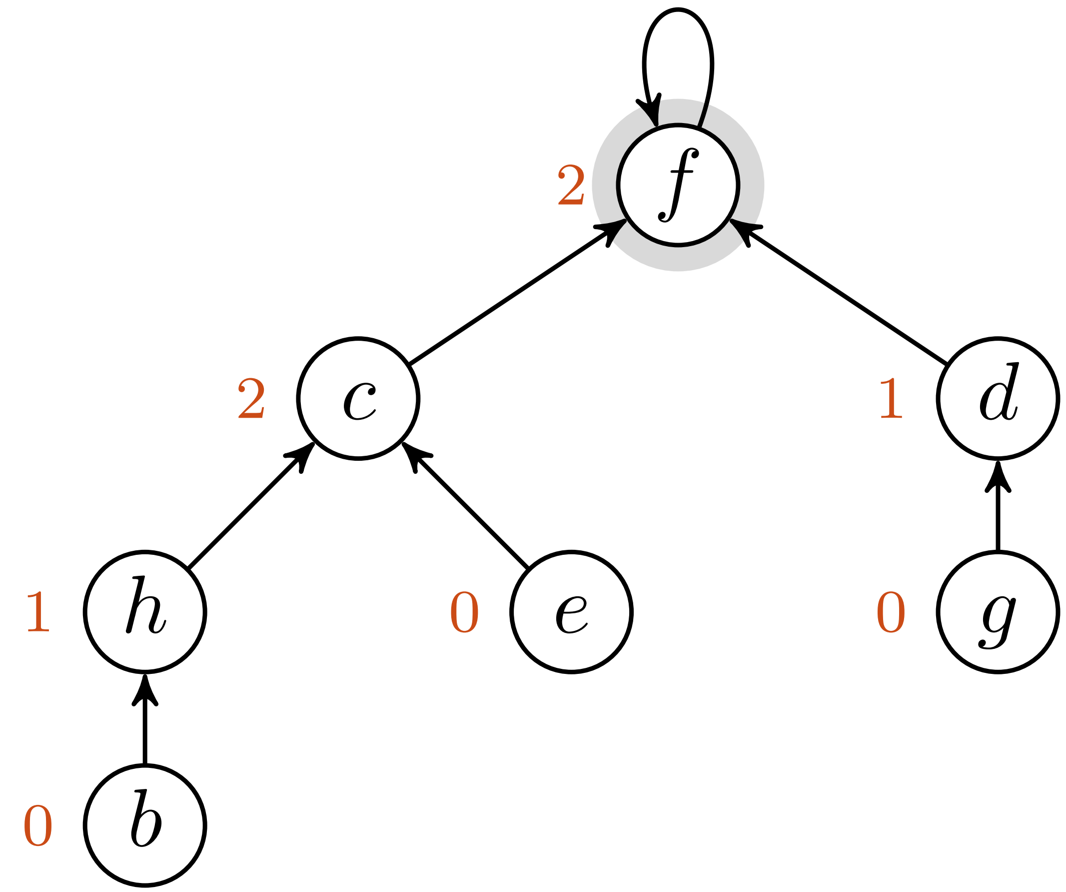
mlh@ntnu.no



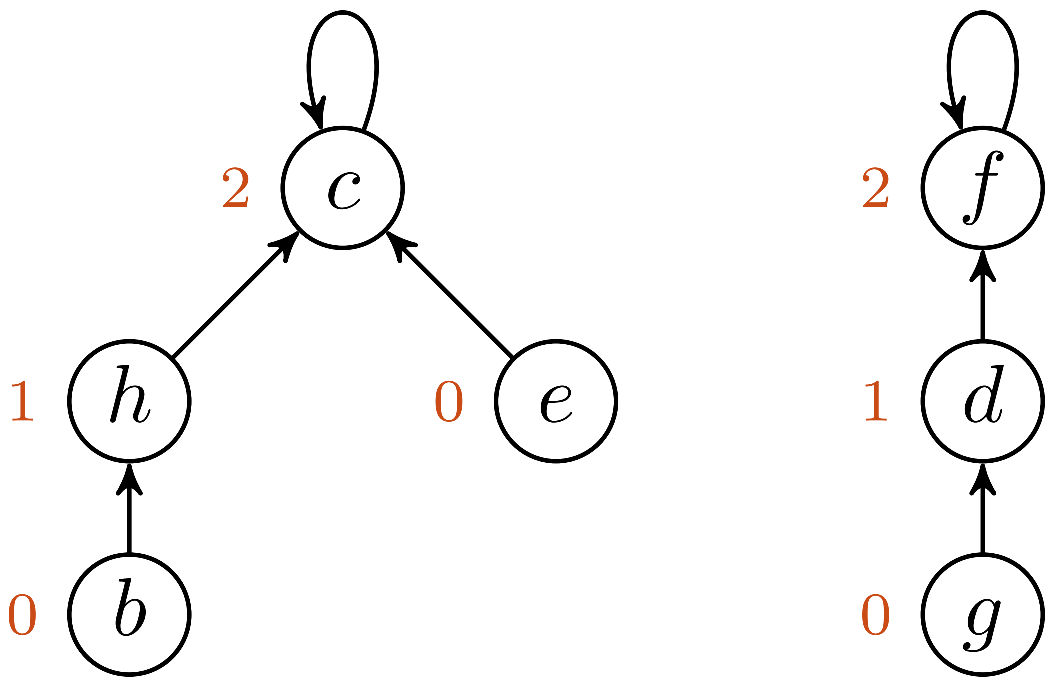


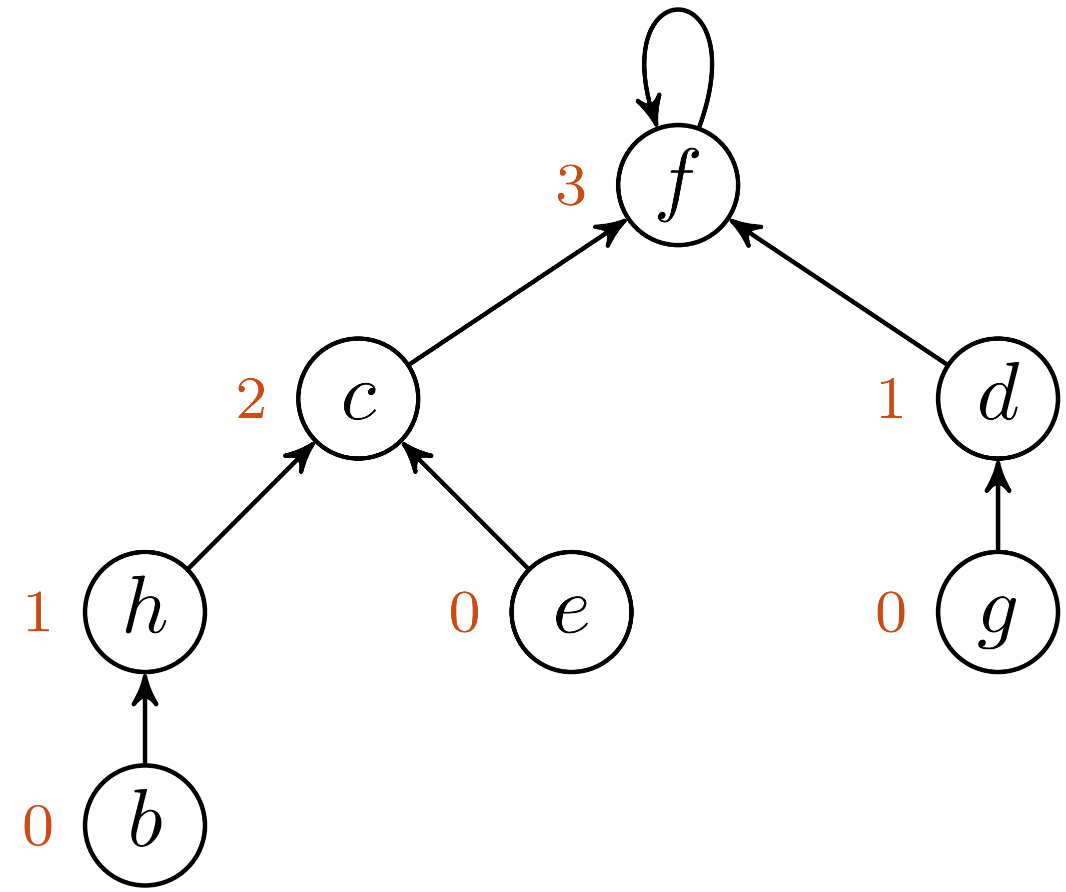
mlh@ntnu.no





mlh@ntnu.no





mlh@ntnu.no

Find-Set(x)

Finn «representanten»/rota til mengden/treet som inneholder x

mlh@ntnu.no

Find-Set(x)
1 **if** $x \neq x.p$

Rota har seg selv som representant. Ikke der ennå . . .

mlh@ntnu.no

```
FIND-SET(x)
  if x ≠ x.p
      x.p = FIND-SET(x.p)
```

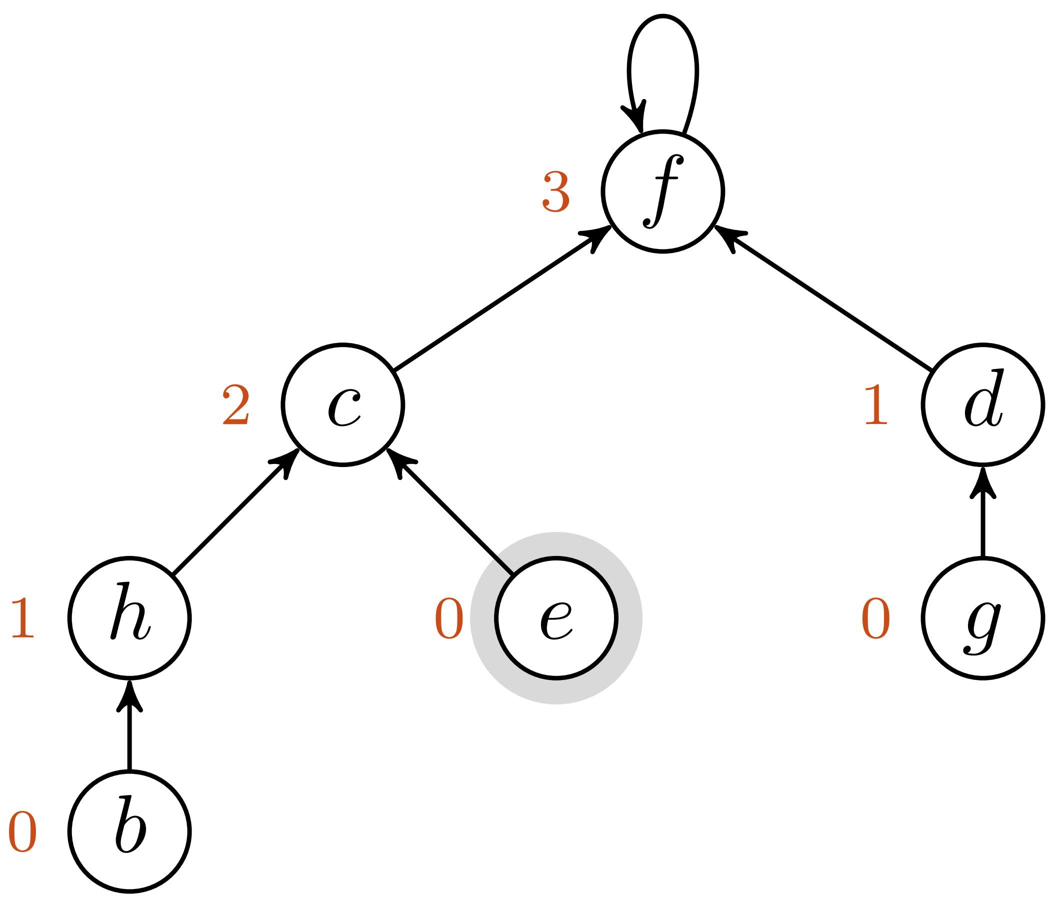
…så vi leter videre (rekursivt)

mlh@ntnu.no

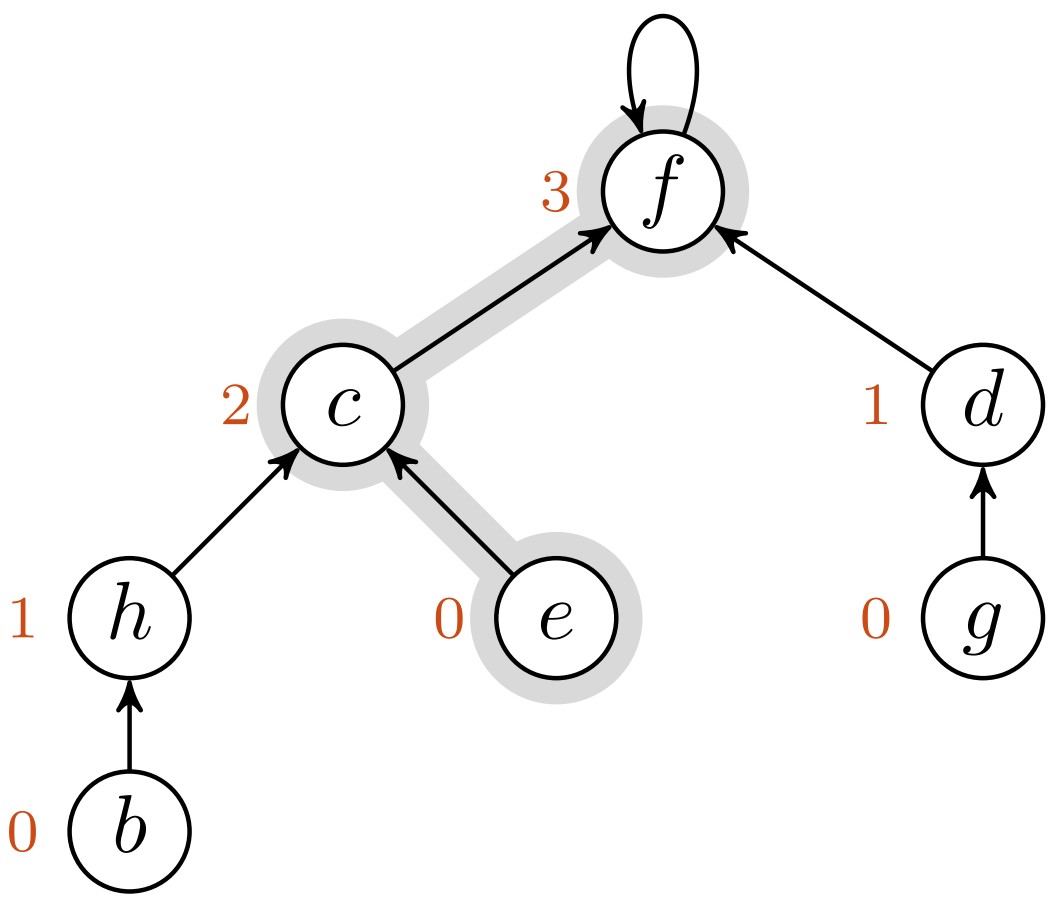
```
FIND-SET(x)
1  if x ≠ x.p
2      x.p = FIND-SET(x.p)
3  return x.p
```

Nå har x fått rota som forelder; snarvei til neste gang!

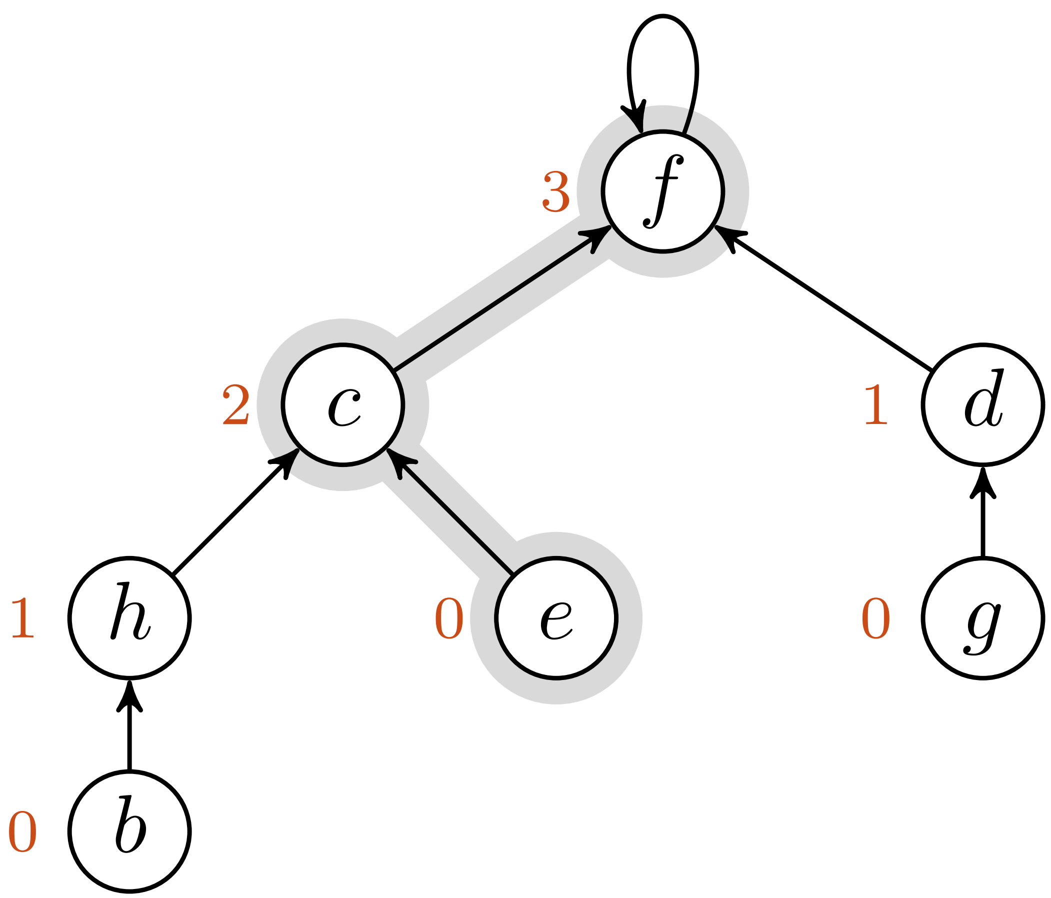
mlh@ntnu.no

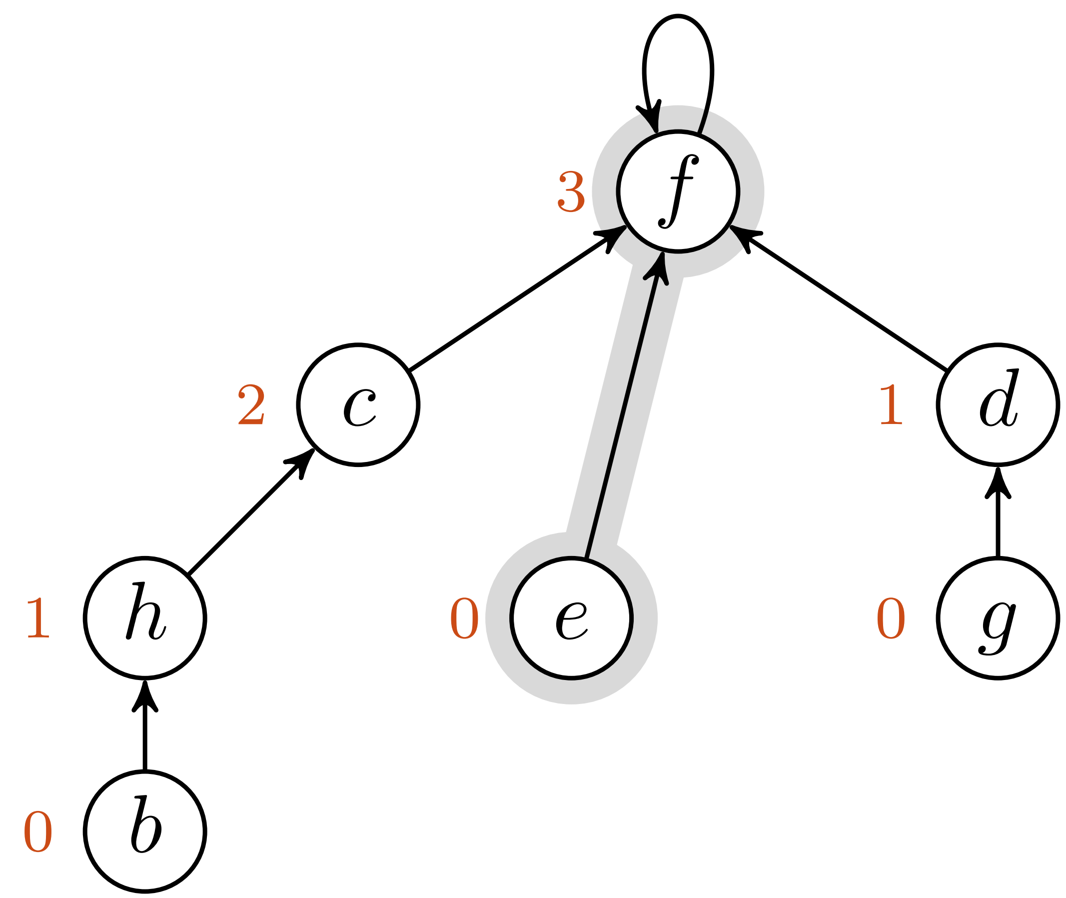


mlh@ntnu.no

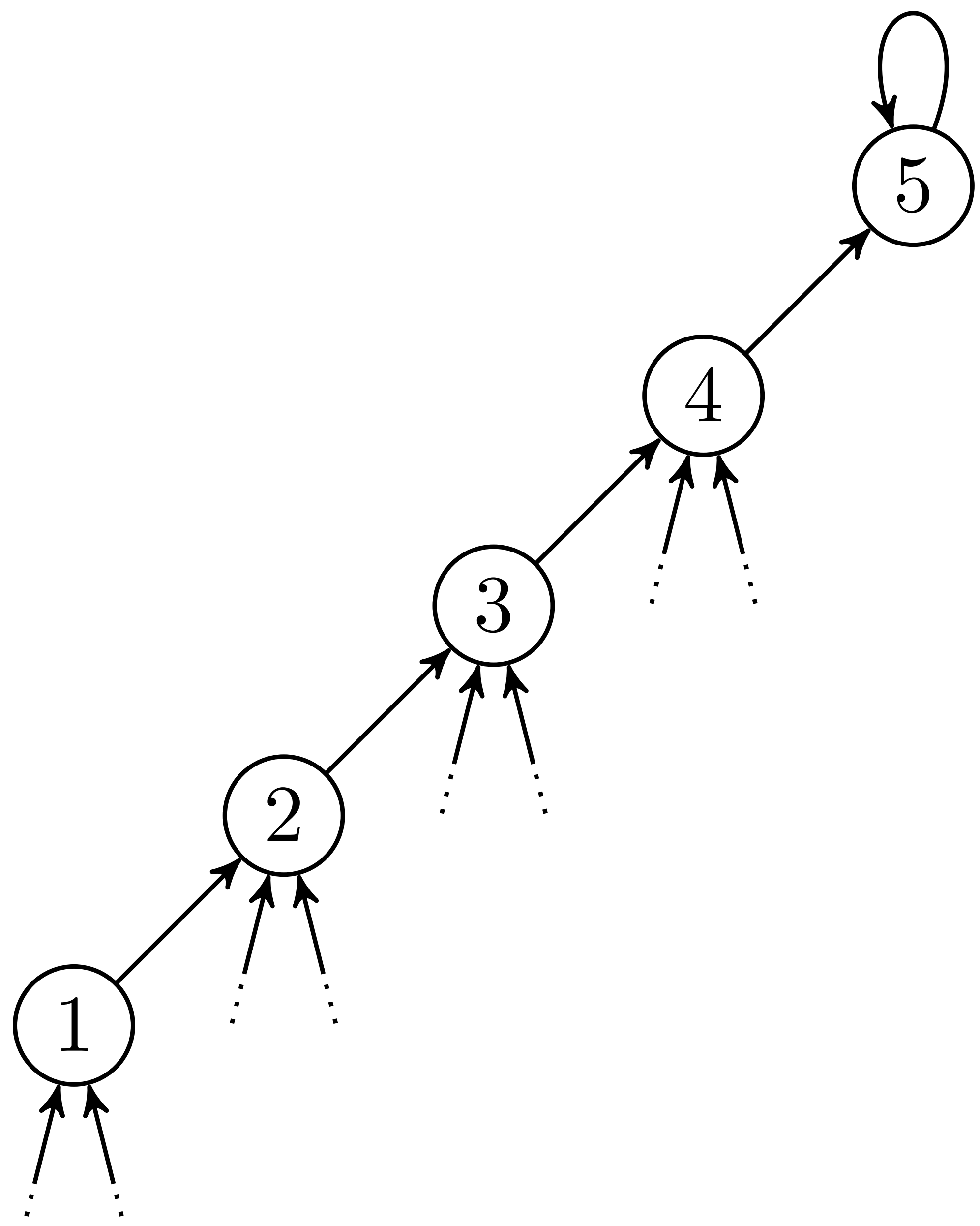


mlh@ntnu.no



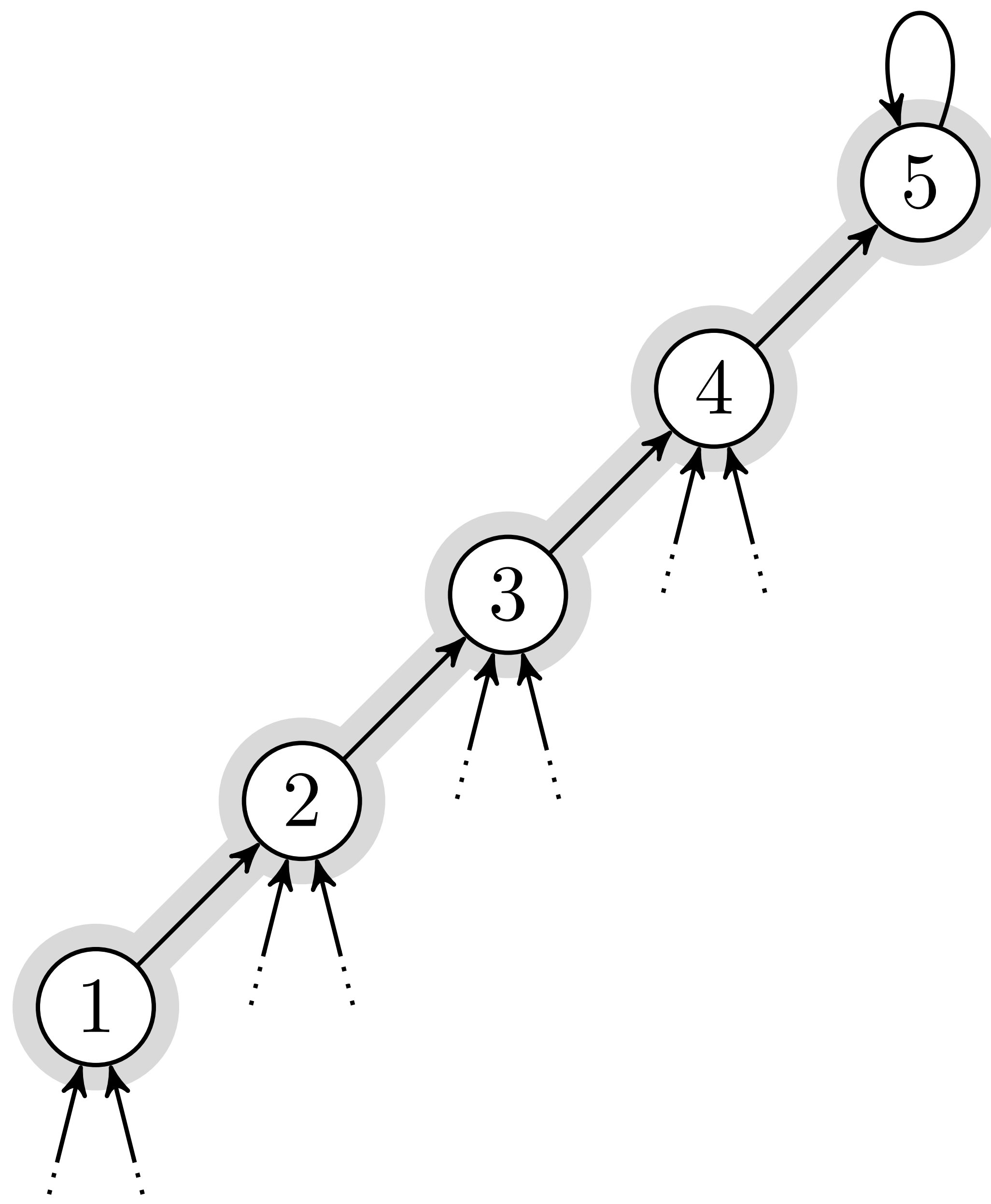


mlh@ntnu.no



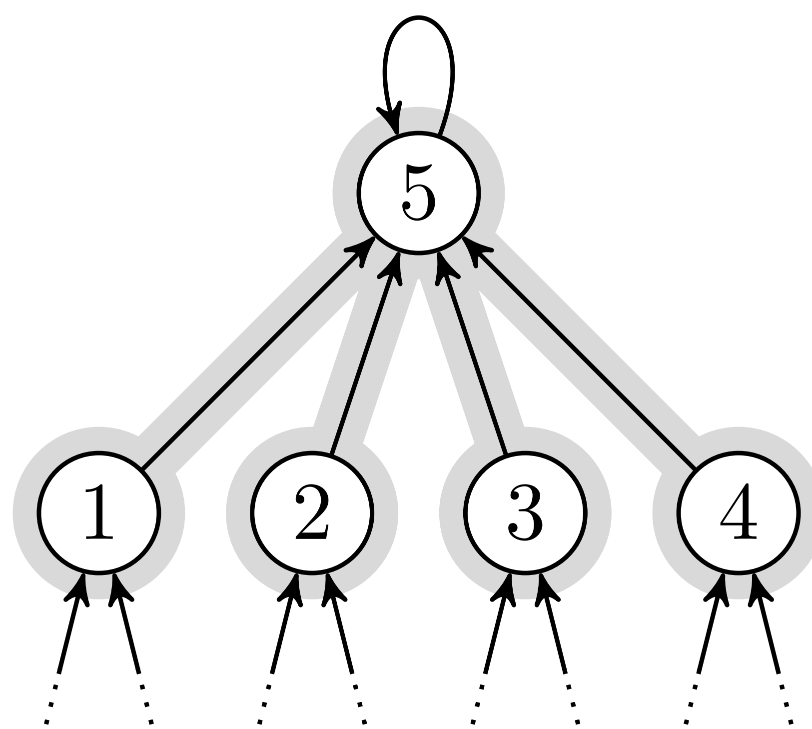
Før FIND-SET(1) komprimerer stien $1 \to \cdots \to 5$

mlh@ntnu.no



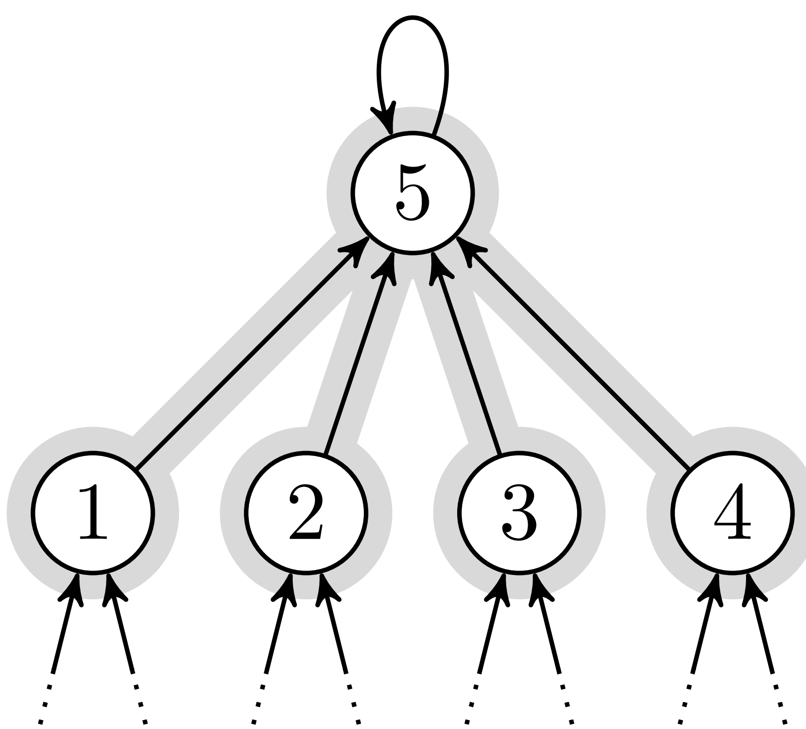
Find-Set er rekursiv og «destruktiv»; alle på stien får $p = 5$

mlh@ntnu.no



FIND-SET er rekursiv og «destruktiv»; alle på stien får $p = 5$

mlh@ntnu.no



Oppdaterer ikke rang; det er derfor det bare er et overestimat

mlh@ntnu.no

m operasjoner: $\mathrm{O}(m \cdot \alpha(n))$

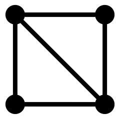
$$\alpha(n) = \begin{cases} 0 & \text{hvis } 0 \leq n \leq 2\,, \\ 1 & \text{hvis } n = 3\,, \\ 2 & \text{hvis } 4 \leq n \leq 7\,, \\ 3 & \text{hvis } 8 \leq n \leq 2047\,, \\ 4 & \text{hvis } 2048 \leq n \leq 16^{512}\,. \end{cases}$$

mlh@ntnu.no

2:4

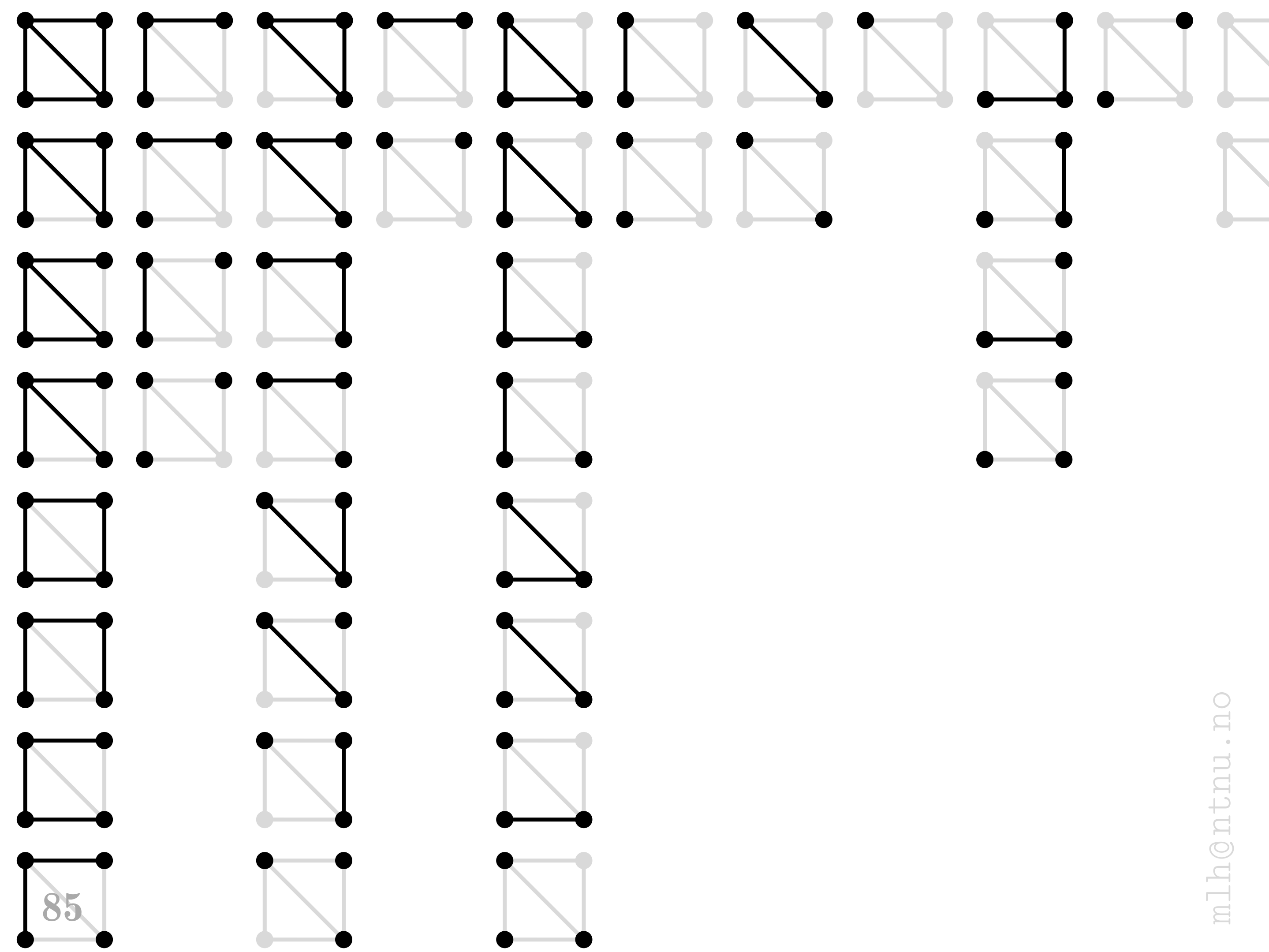
Generisk MST

En graf



mlh@ntnu.no

Delgrafer

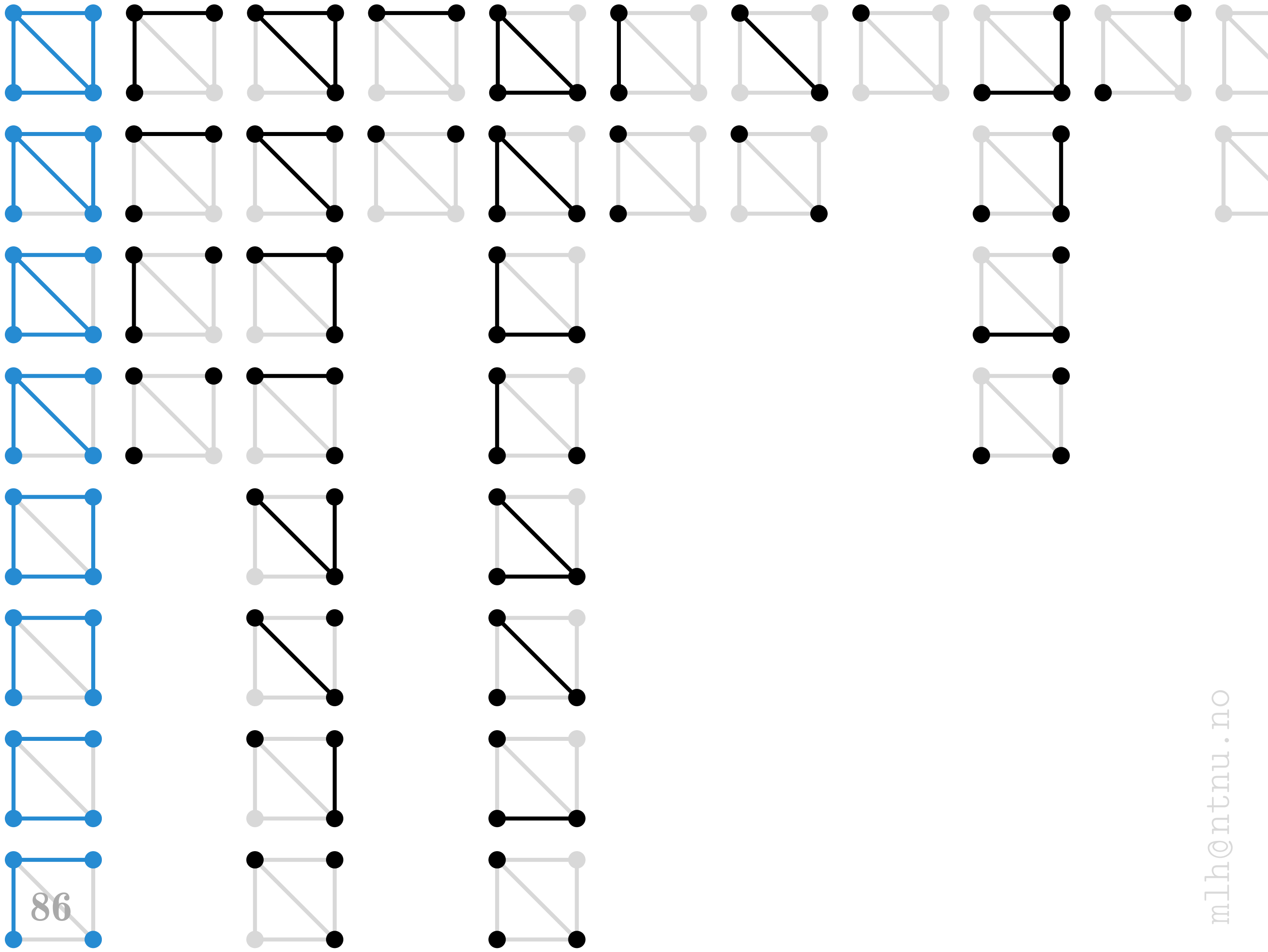


Delgrafer er grafer som består av delmengder av nodene og kantene til den opprinnelige grafen. (En graf er en delgraf av seg selv – men ikke en såkalt *ekte* delgraf.)

mlh@ntnu.no

«Spenngrafer»

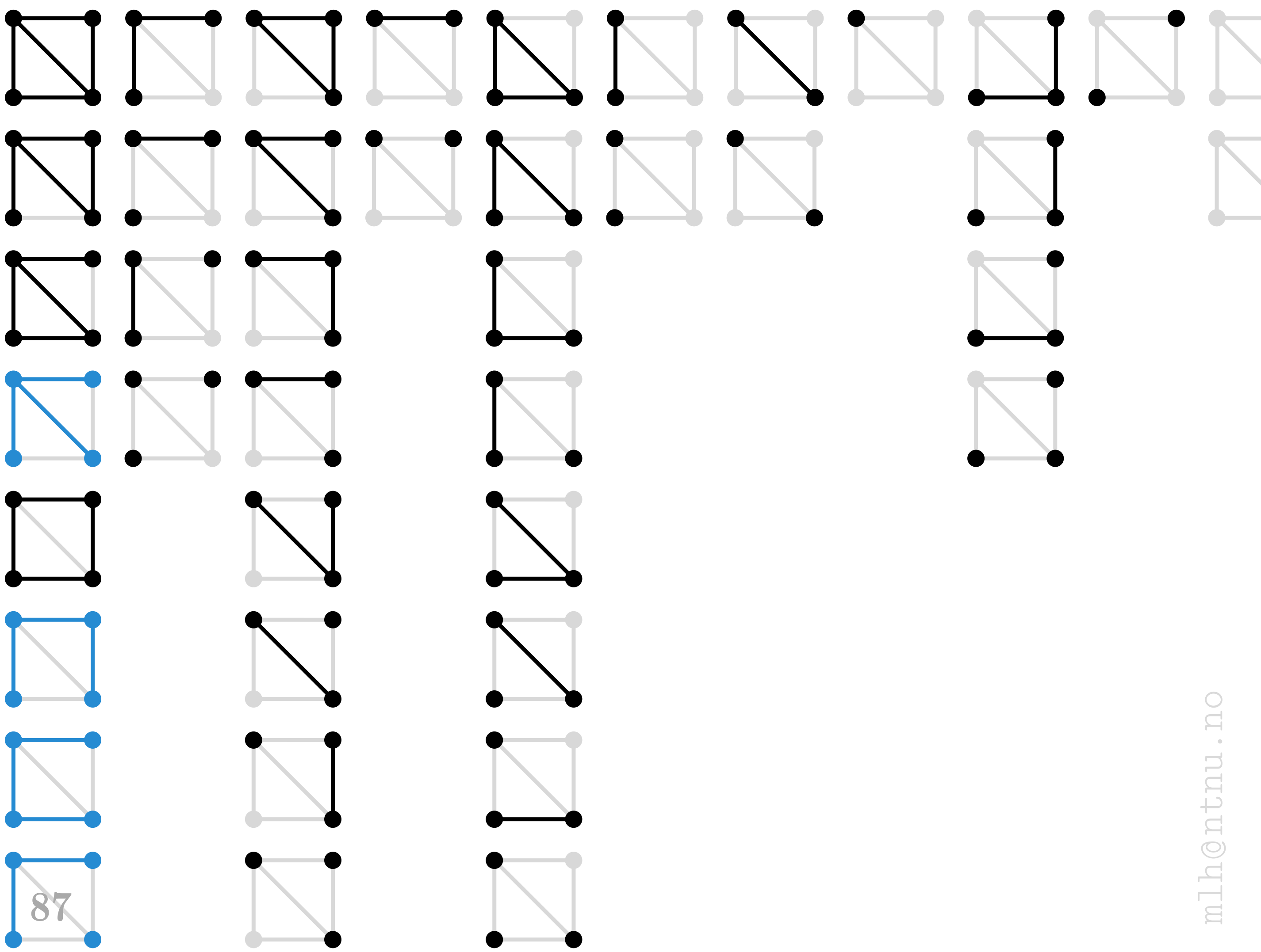
En «dekkende delgraf» (spanning subgraph) eller «spenngraf» er en delgraf med det samme nodesettet som originalgrafen.



mlh@ntnu.no

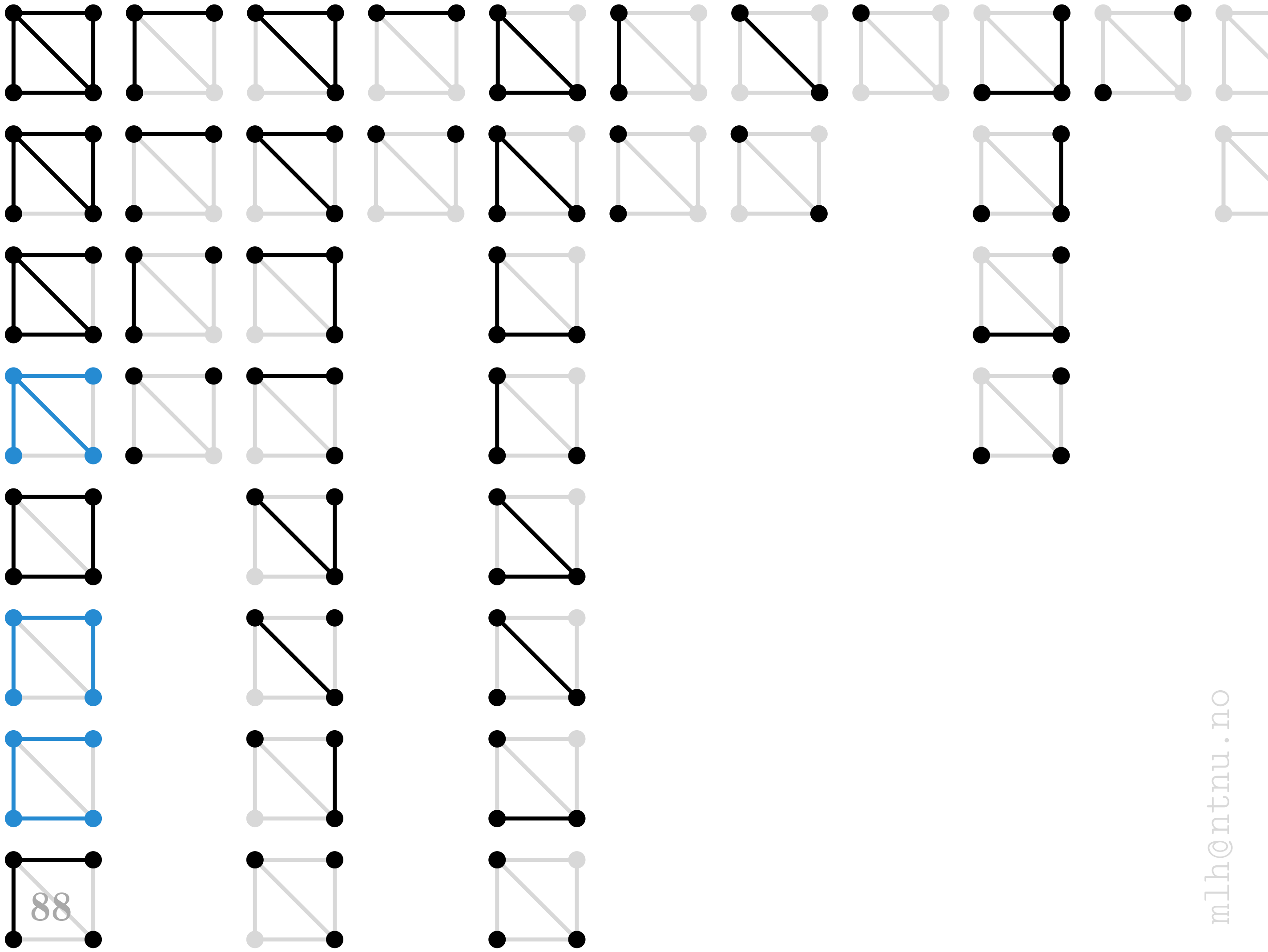
«Spennskoger»

En «dekkende skog» (spanning forest) eler «spenngskog» er en asyklisk spenngraf.



mlh@ntnu.no

Spenntrær



Et spenntre (spanning tree) er en sammenhengende spennskog.

(Spenntre er et vanlig begrep på norsk. Spenngraf og spennskog er det ikke.)

mlh@ntnu.no

PRÁCE
MORAVSKÉ PŘÍRODOVĚDECKÉ SPOLEČNOSTI
SVAZEK III., SPIS 3. 1926 SIGNATURA: F 23
BRNO, ČESKOSLOVENSKO.
ACTA SOCIETATIS SCIENTIARUM NATURALIUM MORAVICAE.
TOMUS III., FASCICULUS 3.; SIGNATURA: F 23; BRNO, ČECHOSLOVAKIA; 1926.

Dr. OTAKAR BORŮVKA:

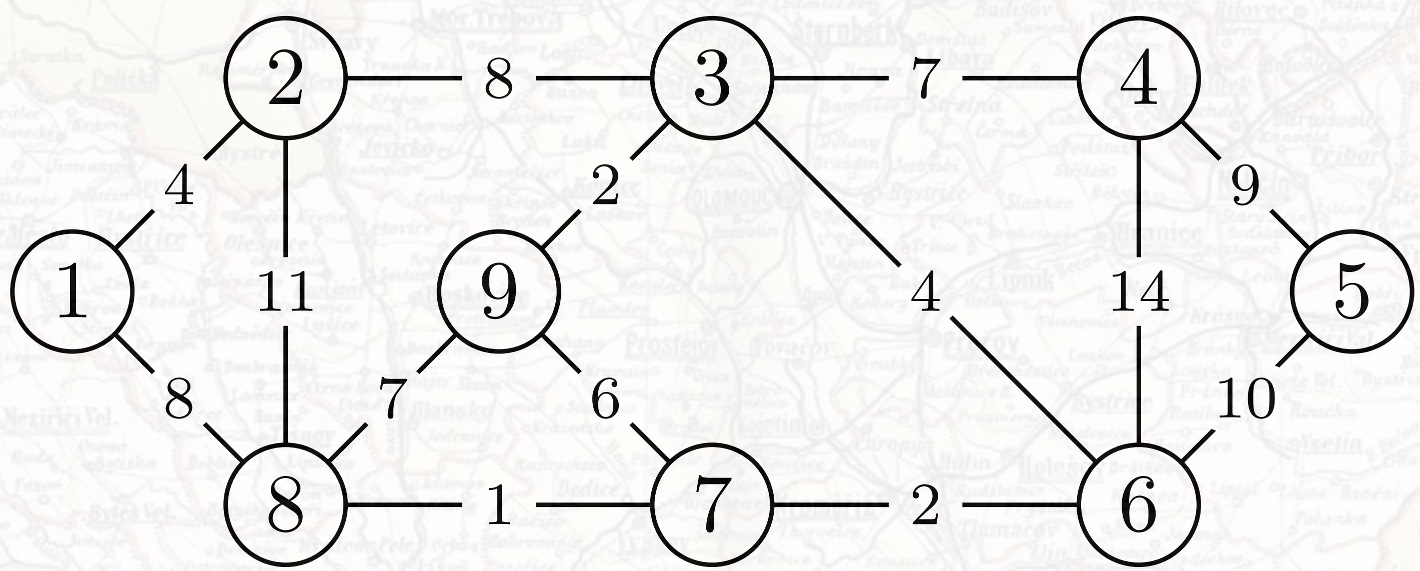
O jistém problému minimálním.

Otakar Borůvka jobbet med å lage billige strømnett i Moravia, og publiserte sin algoritme i 1926.

Vojtěc Jarník bygget i 1930 videre på hans resultater, og konstruerte det som senere oftest har blitt kalt Prims algoritme.

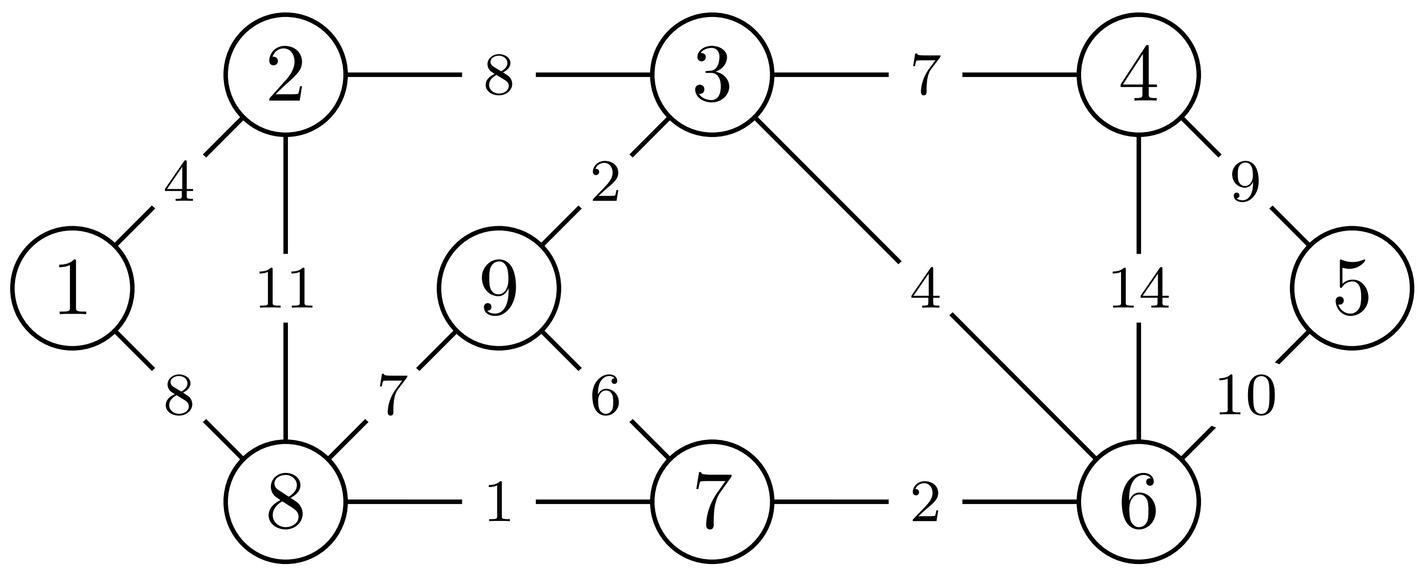
https://commons.wikimedia.org/wiki/

Vi innfører nå <u>vekter</u> på kantene. Disse omtales også som <u>lengder</u> eller <u>kostnader</u>.



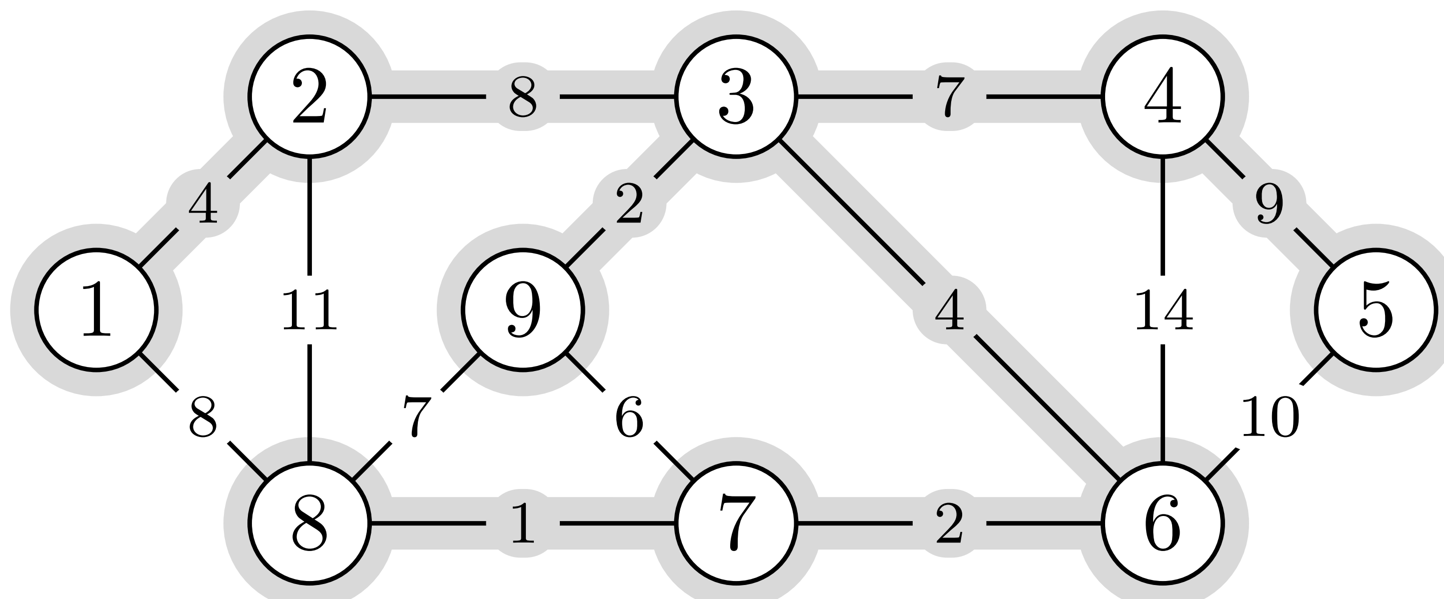
Vi vil knytte sammen nodene billigst mulig

mlh@ntnu.no



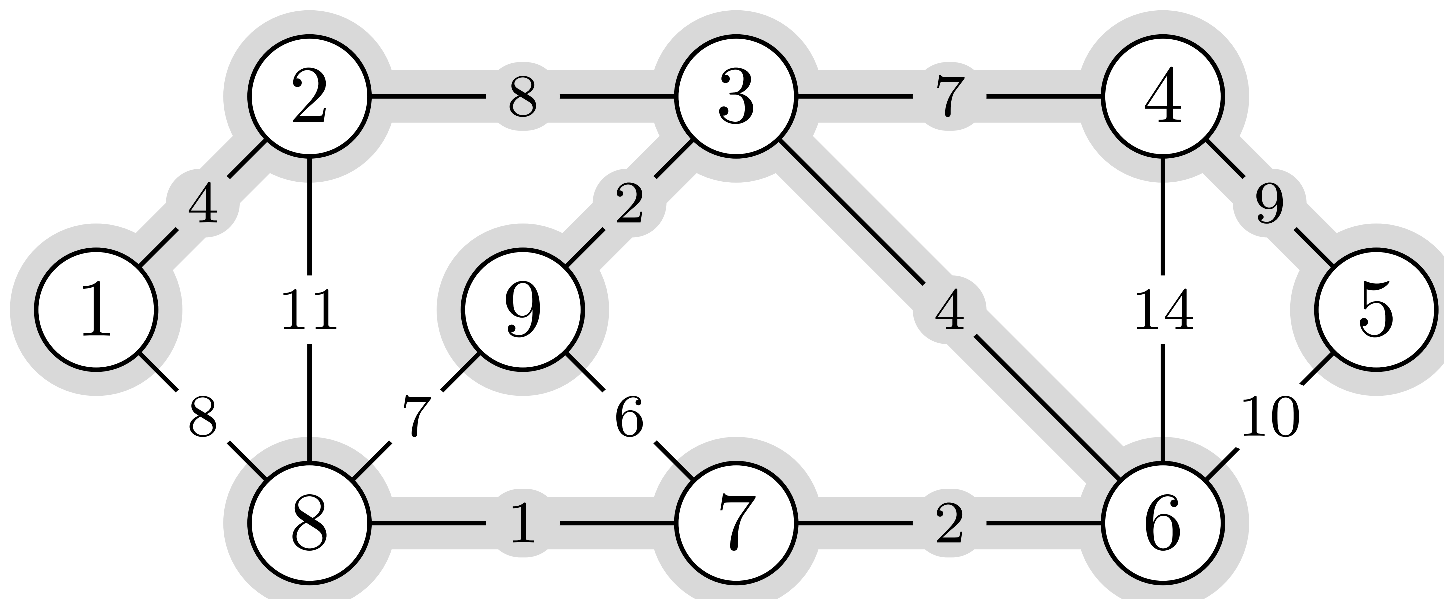
Delmengde $\mathrm{T} \subseteq \mathrm{E}$ som spenner over V og minimerer $\sum_{e \in \mathrm{T}} w(e)$

mlh@ntnu.no



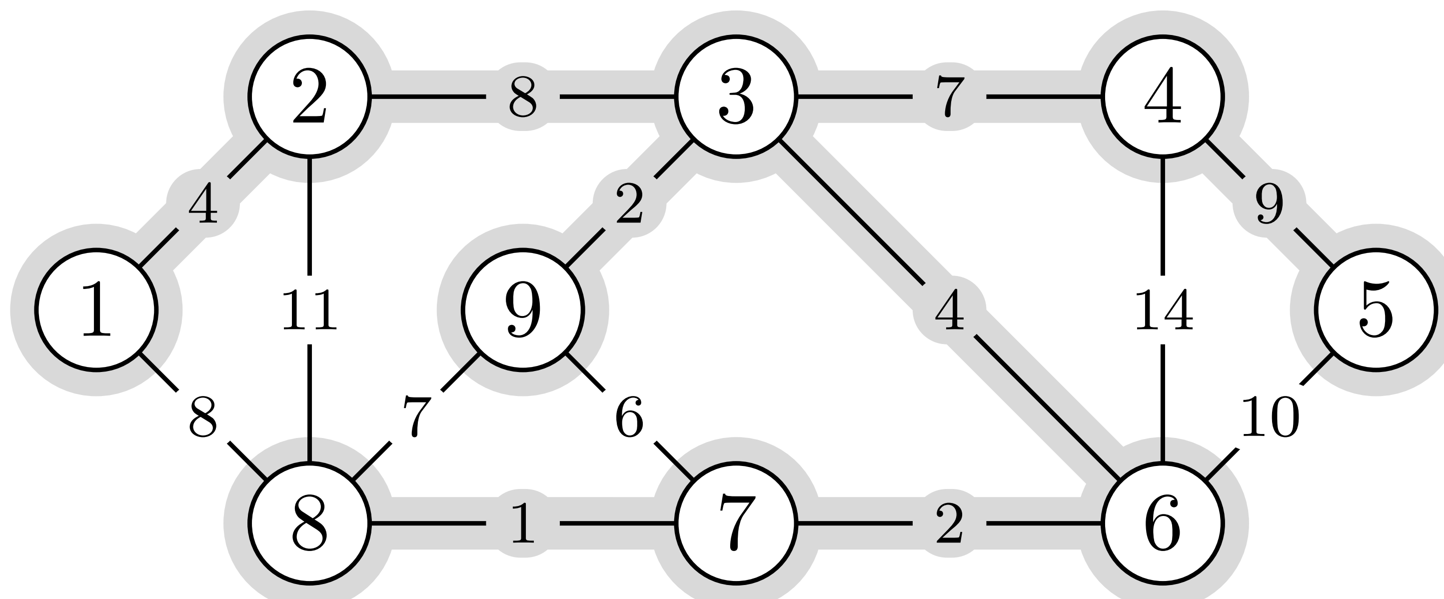
Delmengde $\mathrm{T} \subseteq \mathrm{E}$ som spenner over V og minimerer $\sum_{e \in \mathrm{T}} w(e)$

mlh@ntnu.no



Hvis w er positiv vil T alltid bli asyklisk

mlh@ntnu.no



Generelt: Tillater negativ w men krever asyklisk T

mlh@ntnu.no

Input: En urettet graf $G = (V, E)$ og en vekt-funksjon $w : E \to \mathbb{R}$.

Output: En asyklisk delmengde $T \subseteq E$ som kobler sammen nodene i V og som minimerer vektsummen

$$w(T) = \sum_{(u,v) \in T} w(u, v)\,.$$

T er altså et spenntre for G

mlh@ntnu.no

- **Vi utvider en kantmengde (partiell løsning) gradvis**
- **Invariant: Kantmengden utgjør en del av et minimalt spenntre**
- **En «trygg kant» er en kant som bevarer invarianten**

$\textsc{Generic-MST}(G, w)$

Grådig fremgangsmåte for å finne minimale spenntrær

mlh@ntnu.no

Generic-MST(G, w)
1 $A = \emptyset$

Kantene vi har valgt; skal utgjøre spenntre til slutt

mlh@ntnu.no

Generic-MST(G, w)

1 $A = \emptyset$

2 **while** A does not form a spanning tree

Terminerer om G er sammenhengende og vi ikke lager sykler

mlh@ntnu.no

Generic-MST(G, w)

$A = \emptyset$

while A does not form a spanning tree

  find an edge (u, v) that is safe for A

Det grådige valget. (Men … hva er trygt?)

mlh@ntnu.no

Generic-MST(G, w)

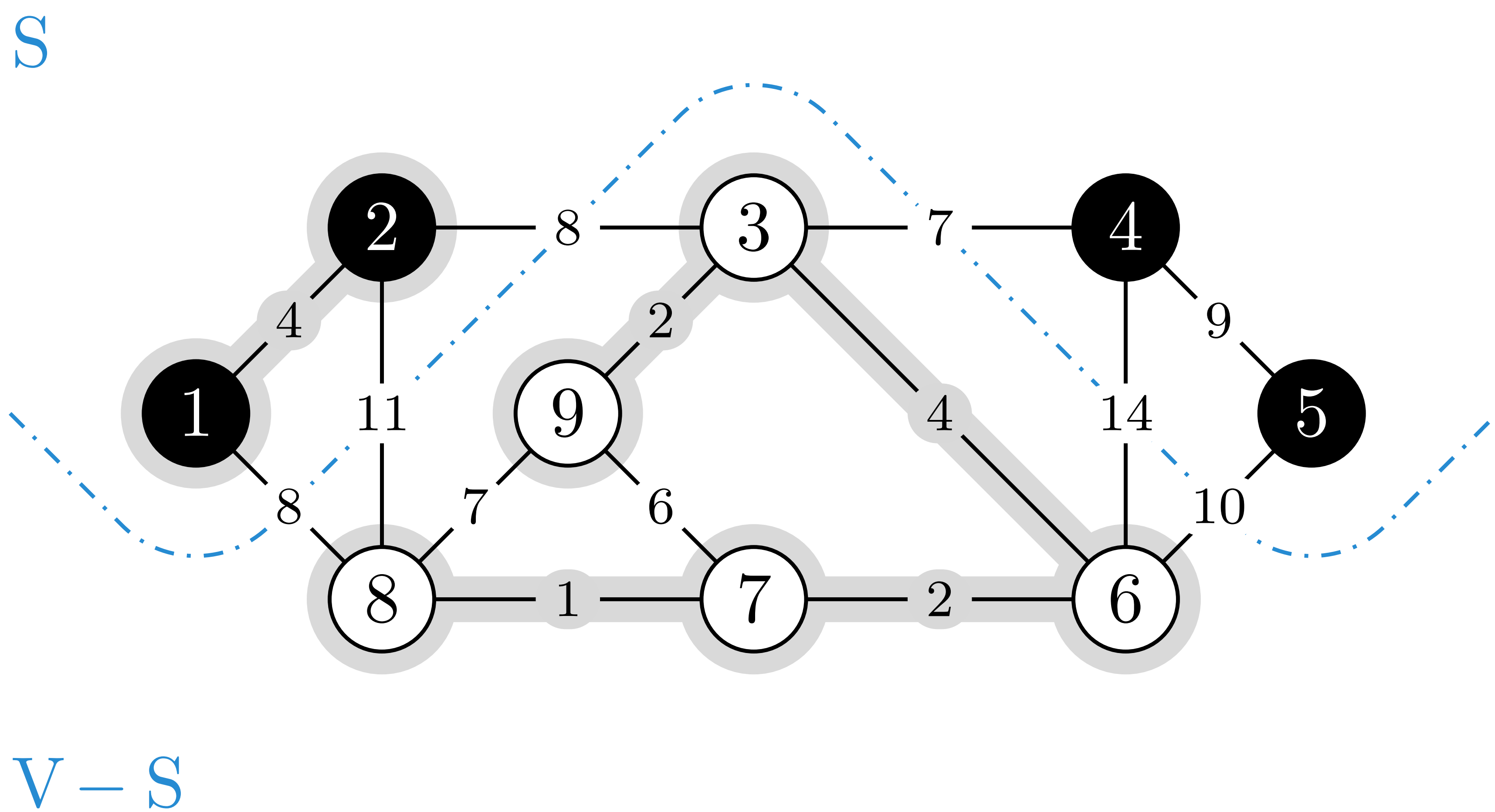
$A = \emptyset$
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    $A = A \cup \{(u, v)\}$

mlh@ntnu.no

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
      find an edge (u, v) that is safe for A
      A = A ∪ {(u, v)}
return A
```


mlh@ntnu.no

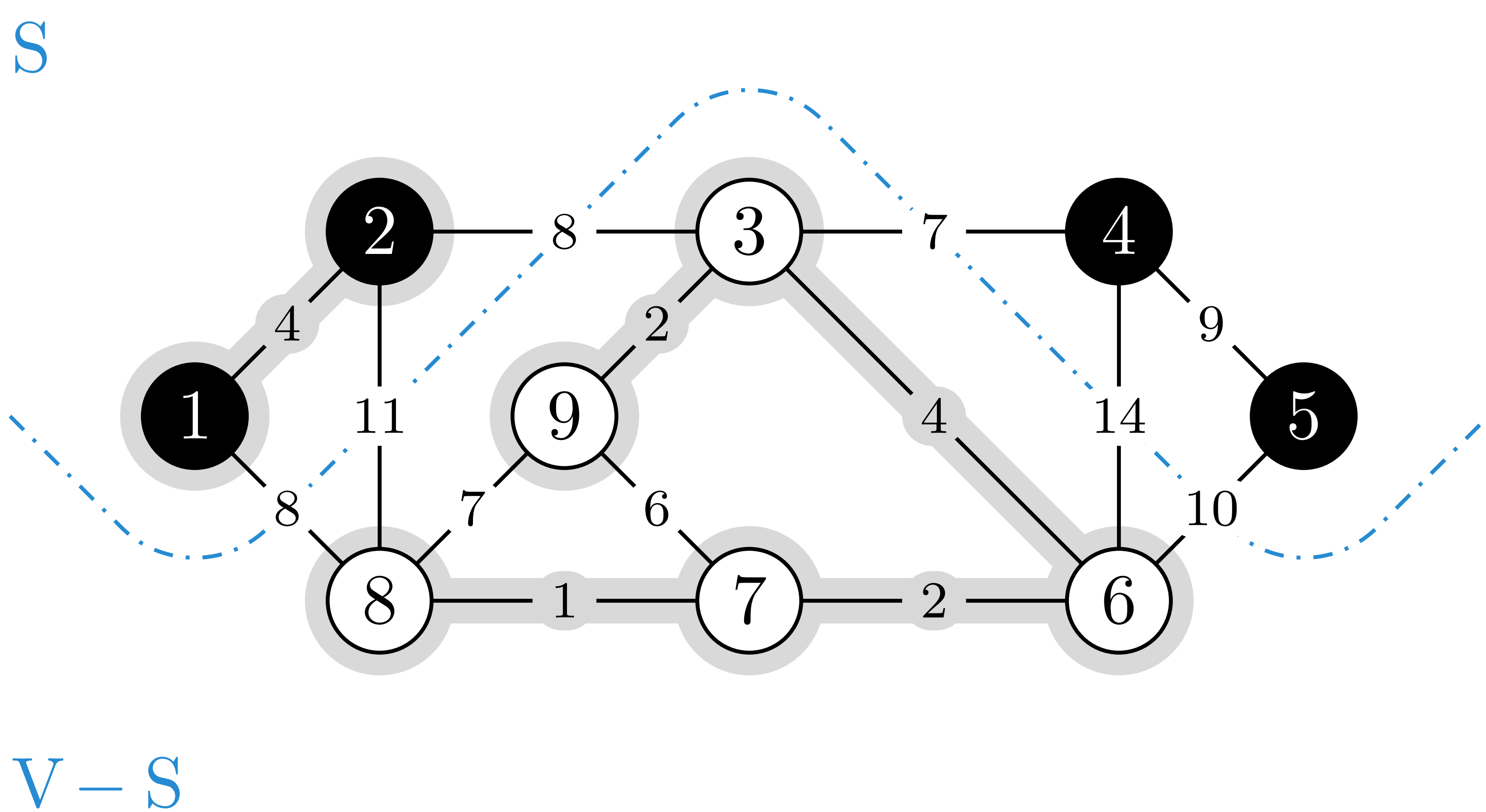
Et snitt er bare en bipartisjon av nodene i grafen. Et snitt respekterer A dersom ingen av kantene i A krysser snittet (dvs., går mellom de to partisjonene).



Et snitt $(S, V - S)$ som respekterer kantmengden A (uthevet)

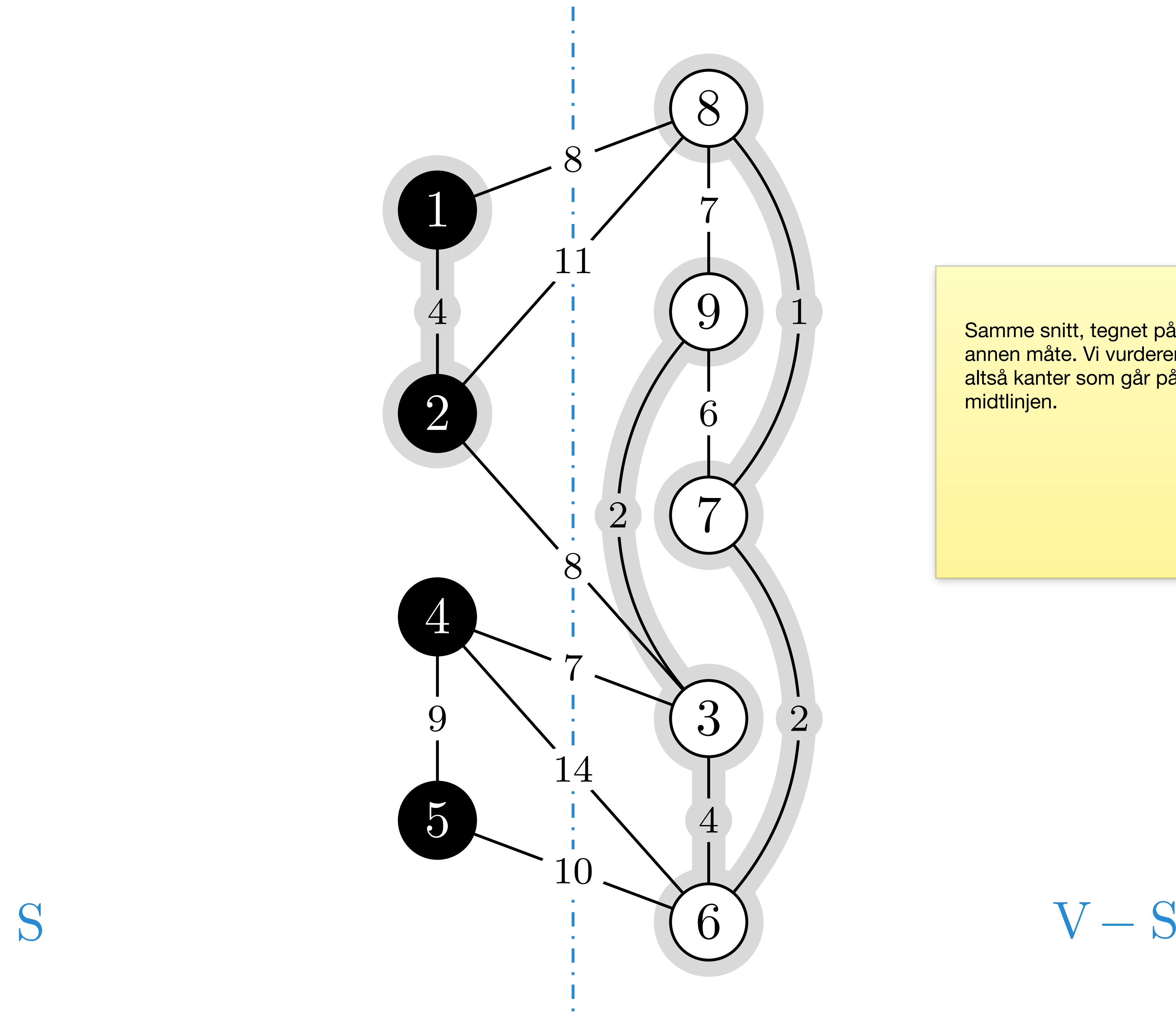
mlh@ntnu.no

En «lett» kant over et snitt er en med minimal vekt blant kantene som går over snittet.



Kanten $(4, 3)$ er en (unik) lett kant over snittet

mlh@ntnu.no

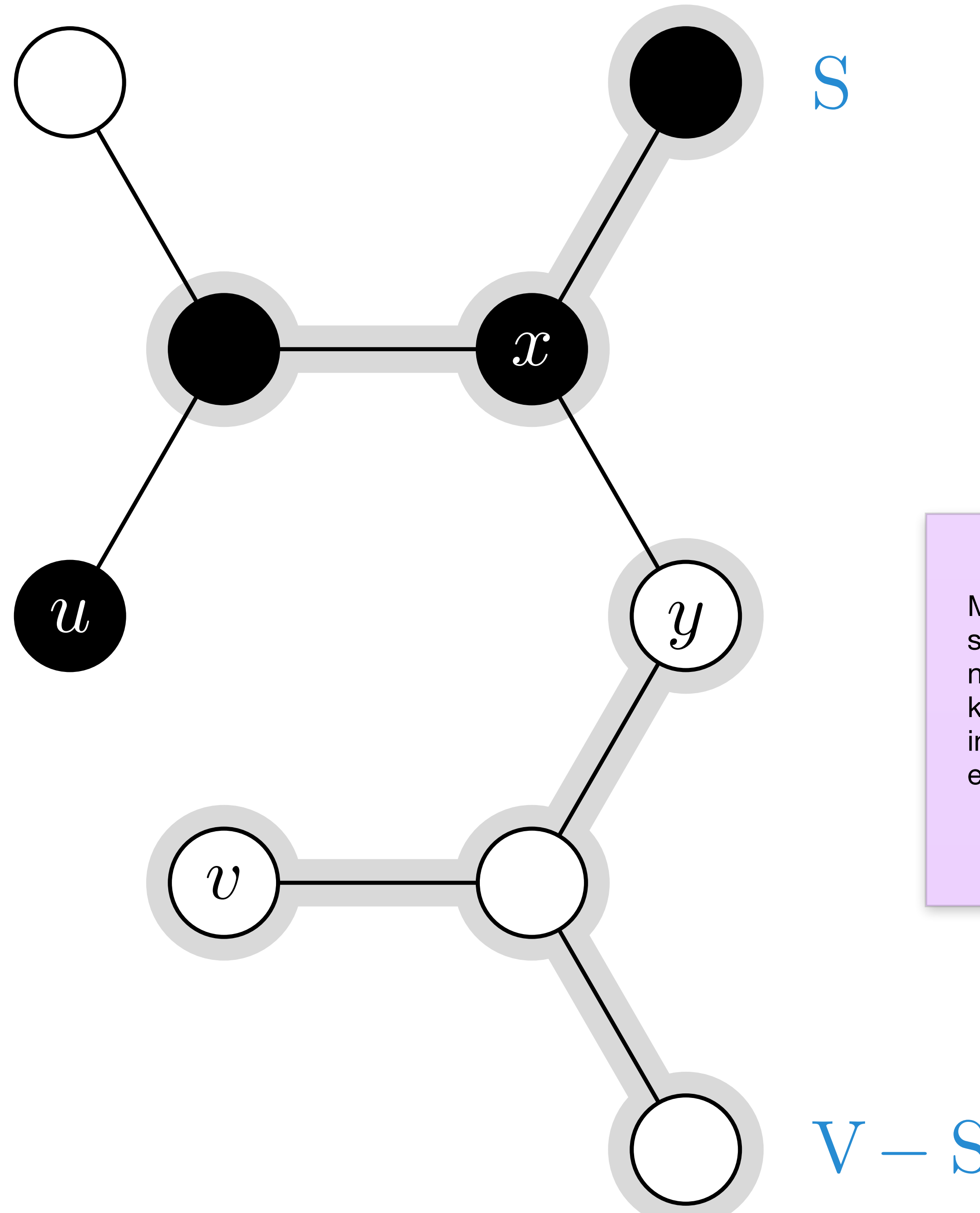


Samme snitt, tegnet på en annen måte. Vi vurderer her altså kanter som går på tvers av midtlinjen.

mlh@ntnu.no

- **«Exchange argument», som før**
- **Ta en optimal (eller vilkårlig) løsning som ikke har valgt grådig**
- **Vis at vi kan endre til det grådige valget uten å få en dårligere løsning**

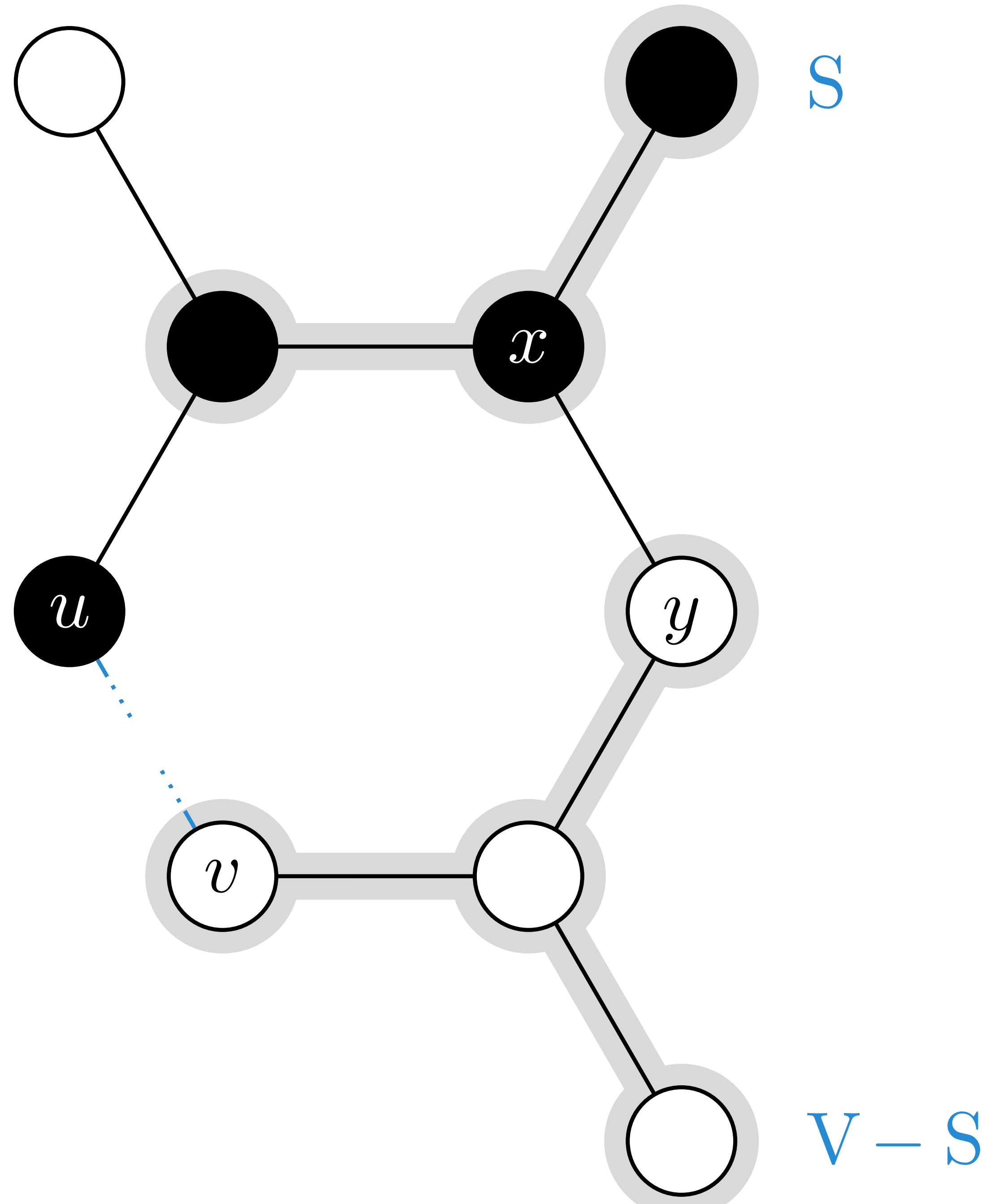
T er den delen av grafen som er tegnet opp her, altså de nodene og kantene vi ser. (Resten av grafen er skjult.)



Mengden S er de svarte nodene, så snittet går mellom de svarte og hvite nodene. Mengden A er de uthevede kantene (som altså respekterer snittet; ingen kanter går mellom en svart og en hvit node.)

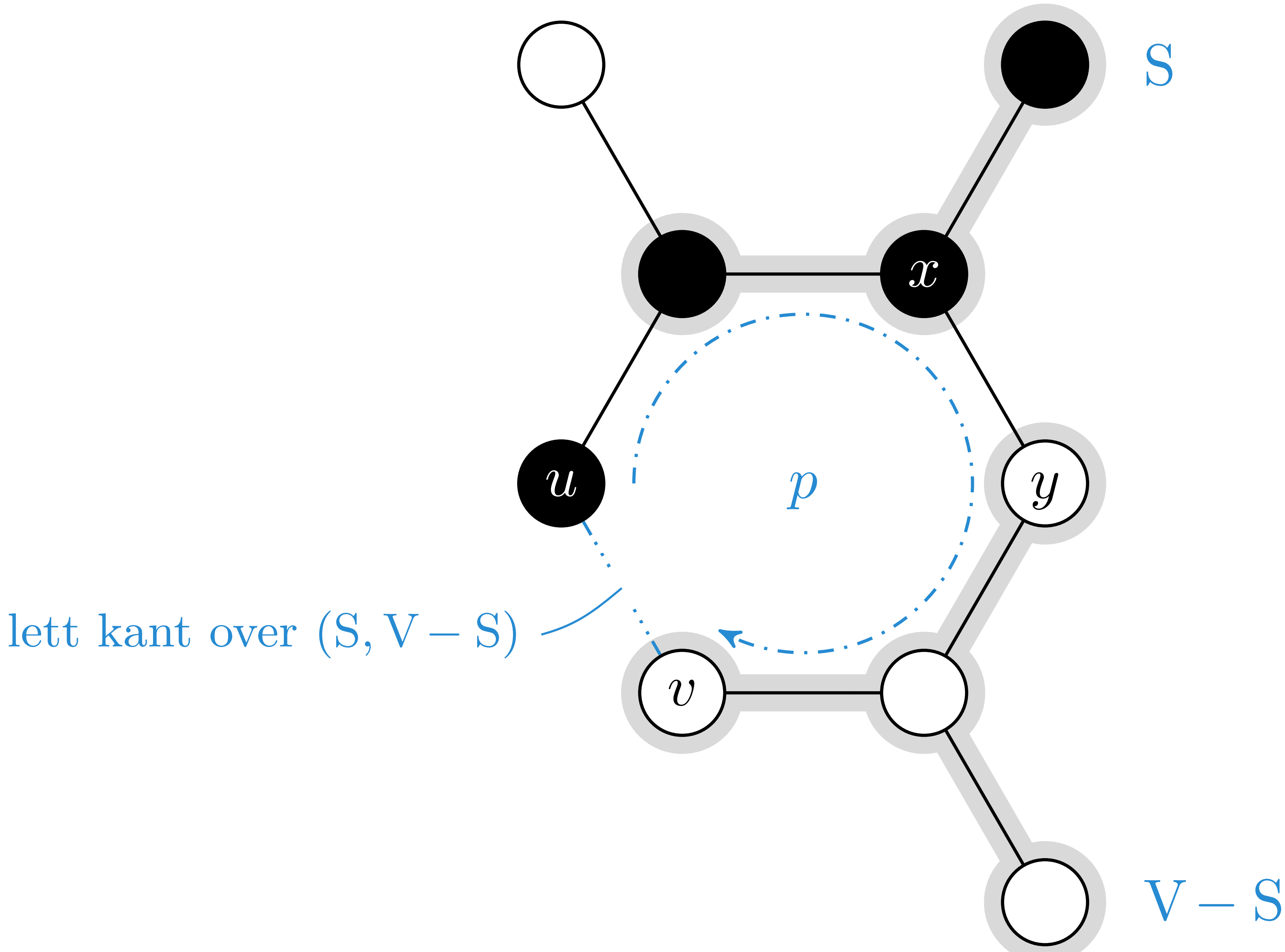
Spenntreet T inneholder A og snittet $(S, V - S)$ respekterer A

mlh@ntnu.no



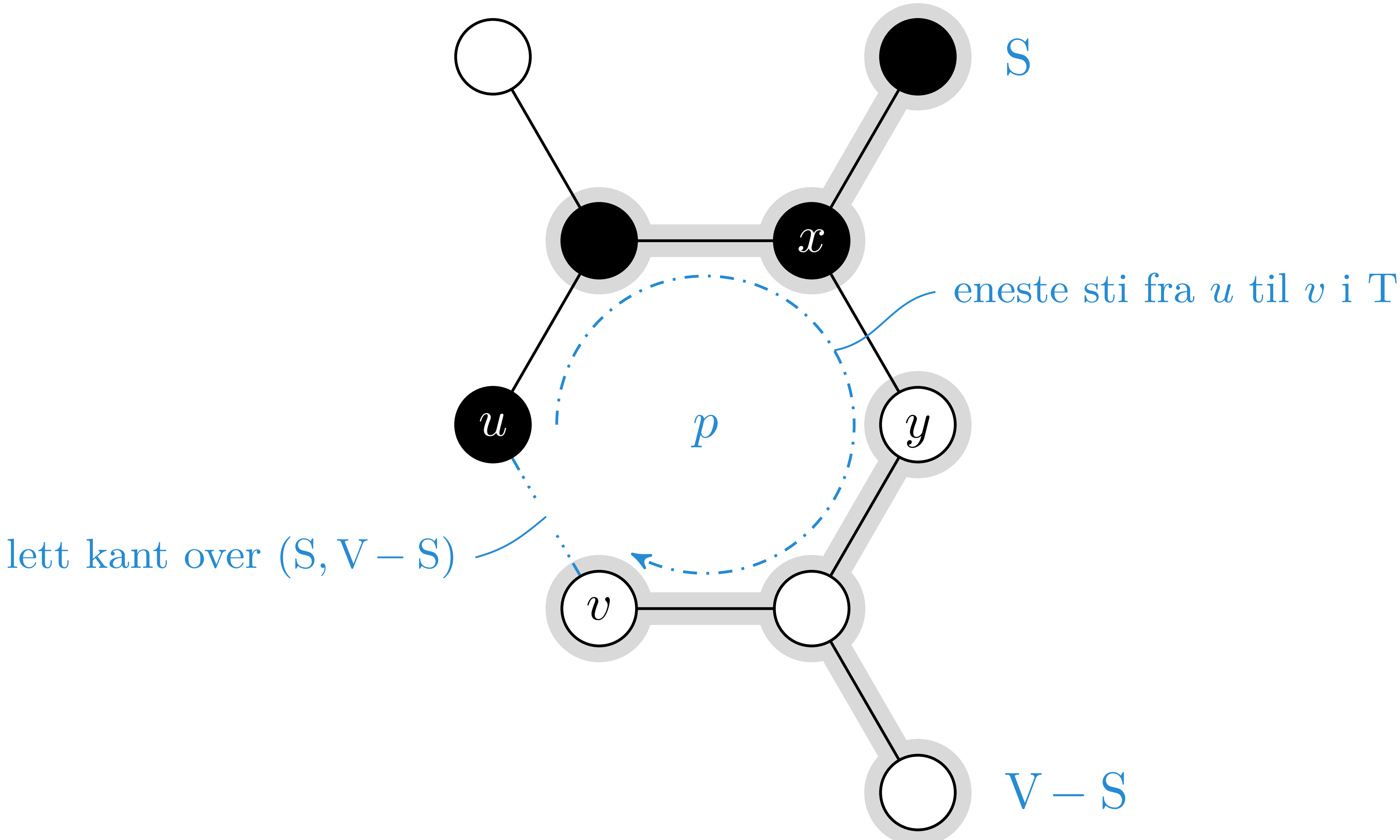
Kanten (u, v) er en lett kant over snittet $(\mathrm{S}, \mathrm{V} - \mathrm{S})$

mlh@ntnu.no



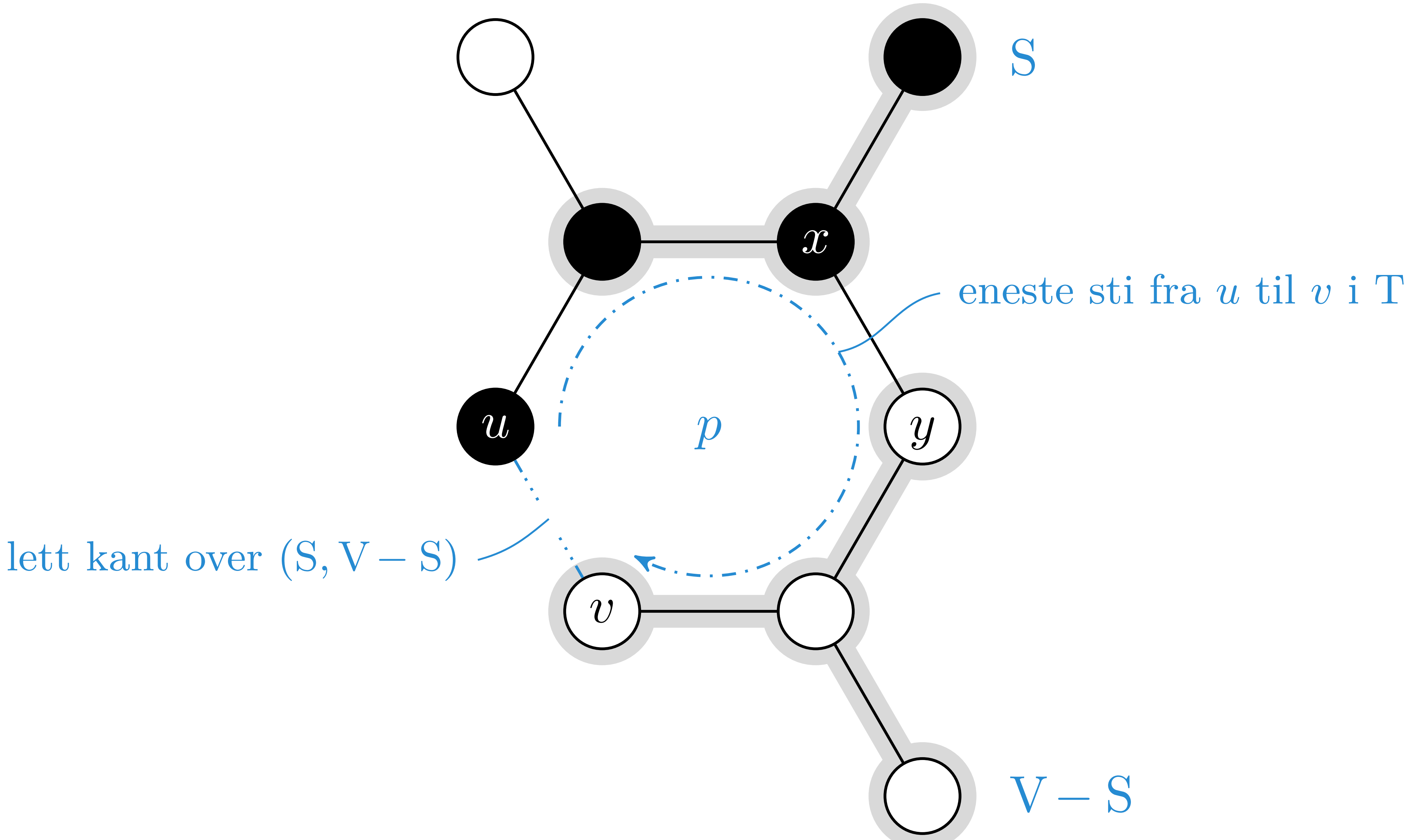
Et spenntre vil ha nøyaktig én sti fra u til v

mlh@ntnu.no



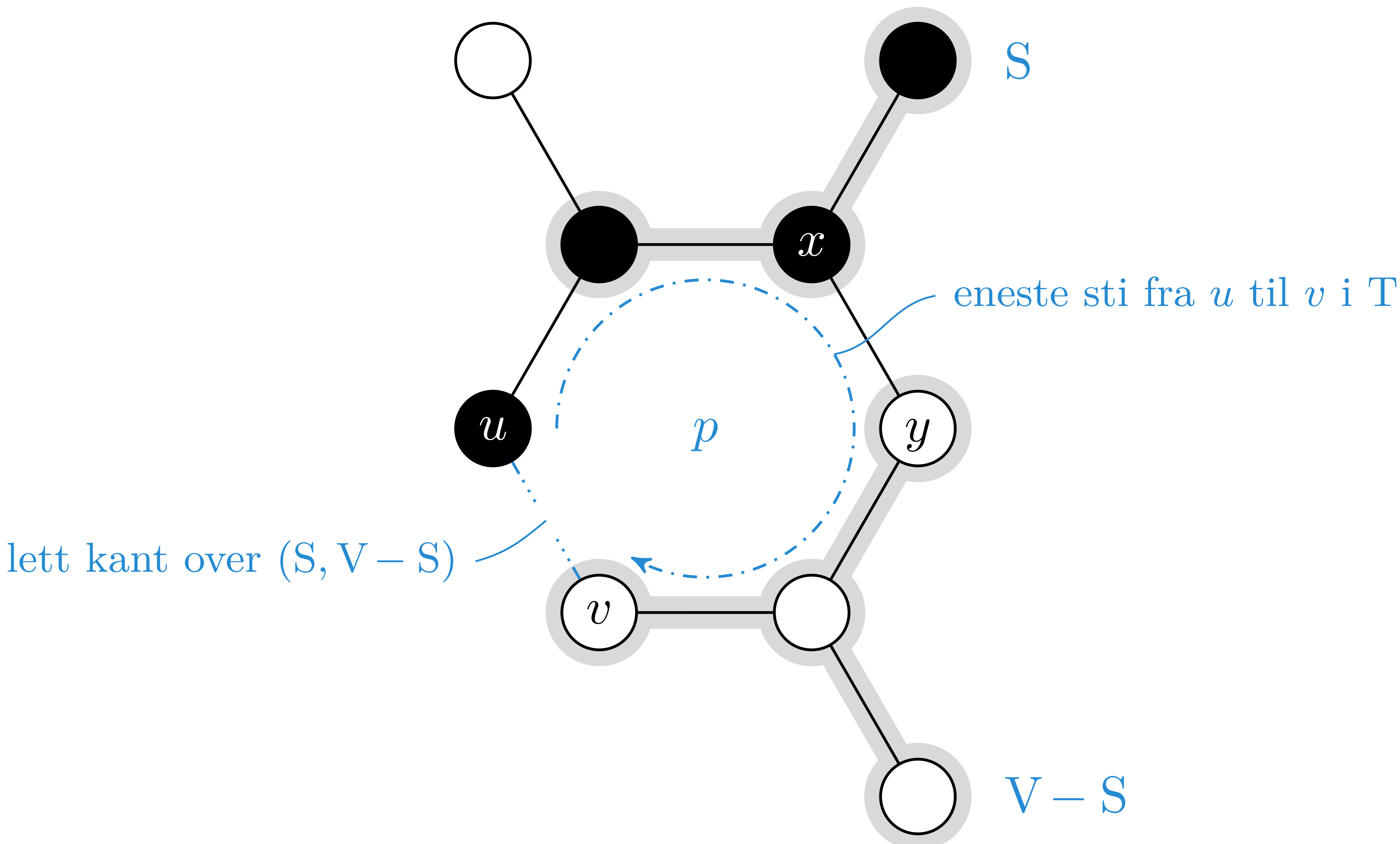
T har én kant (x, y) over snittet, og $w(u, v) \leq w(x, y)$

mlh@ntnu.no



Hvis (u, v) erstatter (x, y) i T', får vi $w(\mathrm{T}') \leq w(\mathrm{T})$

mlh@ntnu.no



T var vilkårlig og T′ minst like bra, så (u, v) er trygg for A

mlh@ntnu.no

Noe å tenke på: De hypotetiske stien hadde her én kant over snittet. Hva om den går i sikk-sakk, og har flere? Holder argumentet fortsatt?

- **En lett kant over et snitt som respekterer løsningen vår er trygg**
- **Vi kan løse problemet grådig!**
- **Men … hvilke snitt skal vi velge?**

Greedy-choice: Det finnes en optimal løsning som inneholder det grådige valget.

Optimal substruktur: Om vi foretar det grådige valget, så må resten løses optimalt (og er av samme type).

Kruskal: En kant med minimal vekt blant de gjenværende er trygg så lenge den ikke danner sykler.

mlh@ntnu.no

Kruskal: En kant med minimal vekt blant de gjenværende er trygg så lenge den ikke danner sykler.

Prim: Bygger ett tre gradvis; en lett kant over snittet rundt treet er alltid trygg.

Prims algoritme ble egentlig først oppfunnet av Jarník

mlh@ntnu.no

Kruskal: En kant med minimal vekt blant de gjenværende er trygg så lenge den ikke danner sykler.

Prim: Bygger ett tre gradvis; en lett kant over snittet rundt treet er alltid trygg.

Borůvka: En slags blanding. Kobler hvert tre til det nærmeste av de andre.

mlh@ntnu.no

Kruskal: En kant med minimal vekt blant de gjenværende er trygg så lenge den ikke danner sykler.

Prim: Bygger ett tre gradvis; en lett kant over snittet rundt treet er alltid trygg.

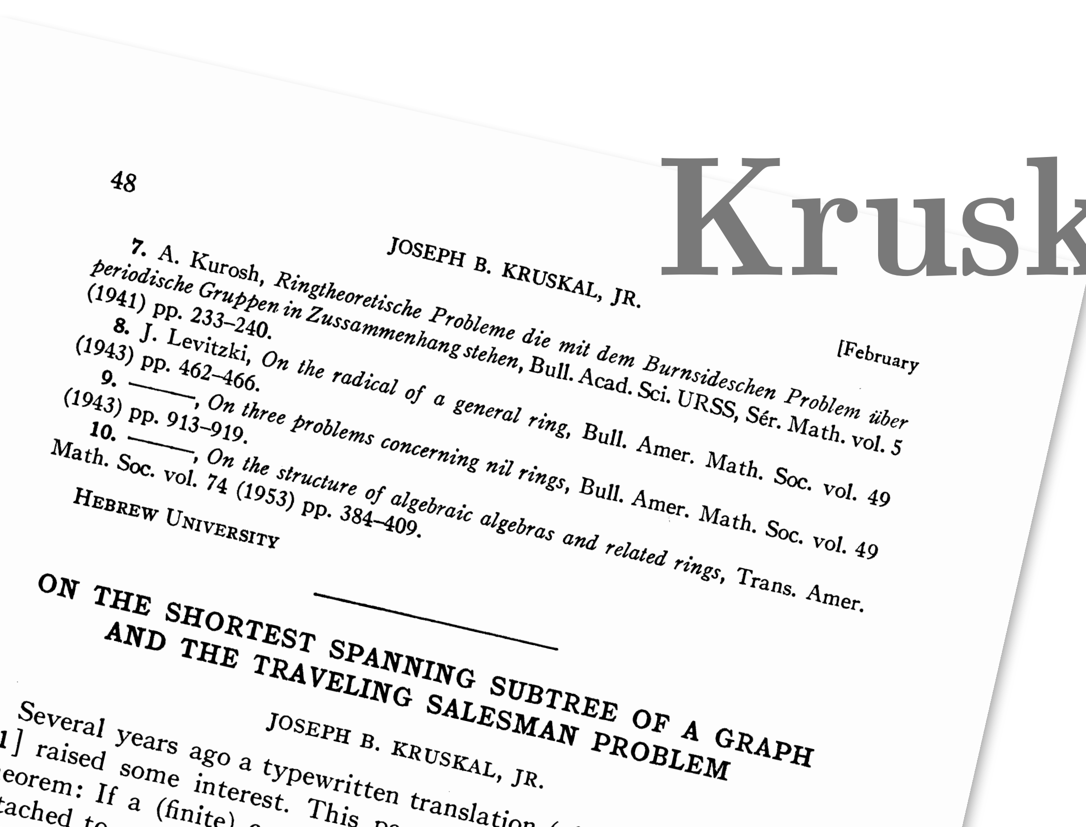
(**Borůvka:** En slags blanding. Kobler hvert tre til det nærmeste av de andre.)

Borůvkas algoritme er ikke sentral; ikke beskrevet i læreboka

mlh@ntnu.no

3:4

Kruskals algoritme


48

JOSEPH B. KRUSKAL, JR. [February

7. A. Kurosh, *Ringtheoretische Probleme die mit dem Burnsideschen Problem über periodische Gruppen in Zussammenhang stehen*, Bull. Acad. Sci. URSS, Sér. Math. vol. 5 (1941) pp. 233–240.
8. J. Levitzki, *On the radical of a general ring*, Bull. Amer. Math. Soc. vol. 49 (1943) pp. 462–466.
9. ———, *On three problems concerning nil rings*, Bull. Amer. Math. Soc. vol. 49 (1943) pp. 913–919.
10. ———, *On the structure of algebraic algebras and related rings*, Trans. Amer. Math. Soc. vol. 74 (1953) pp. 384–409.

HEBREW UNIVERSITY

ON THE SHORTEST SPANNING SUBTREE OF A GRAPH AND THE TRAVELING SALESMAN PROBLEM

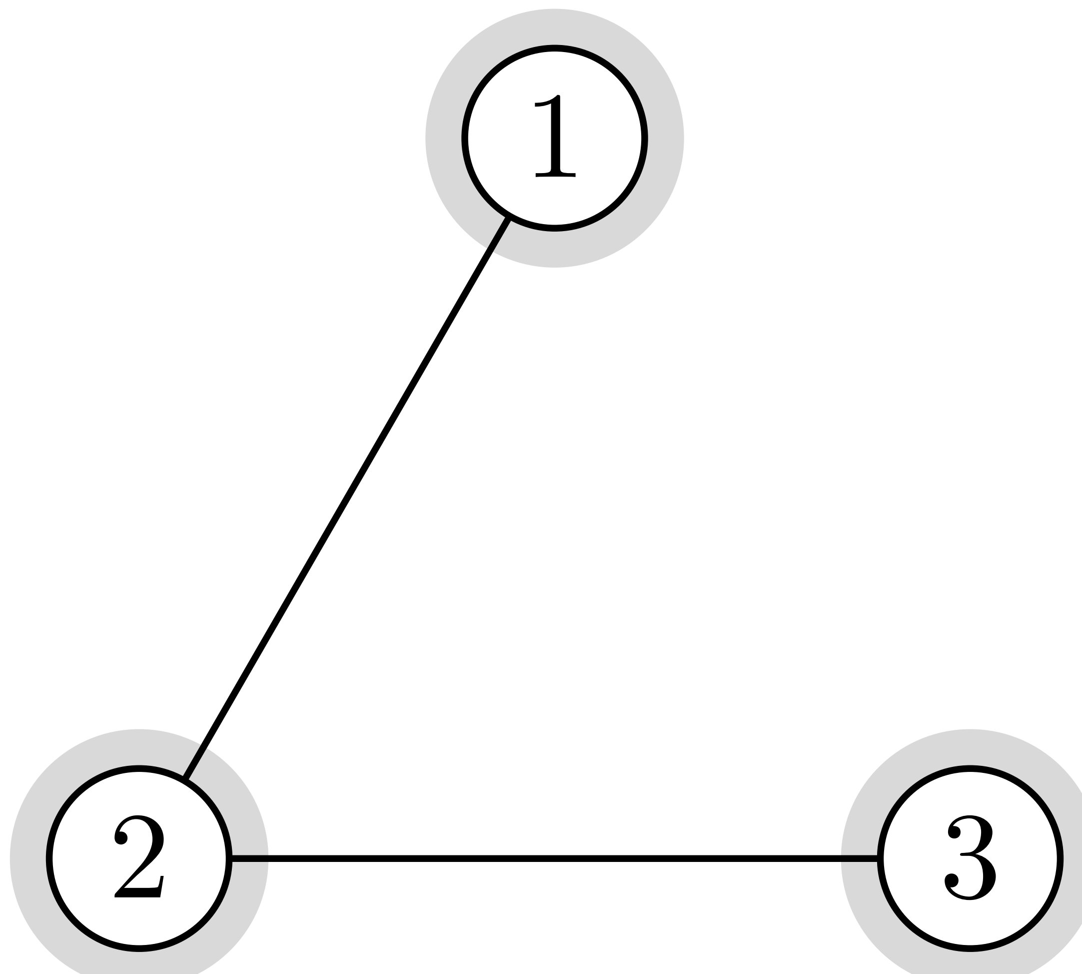
JOSEPH B. KRUSKAL, JR.

Several years ago a typewritten translation
raised some interest. This
eorem: If a (finite)

Fra 1956

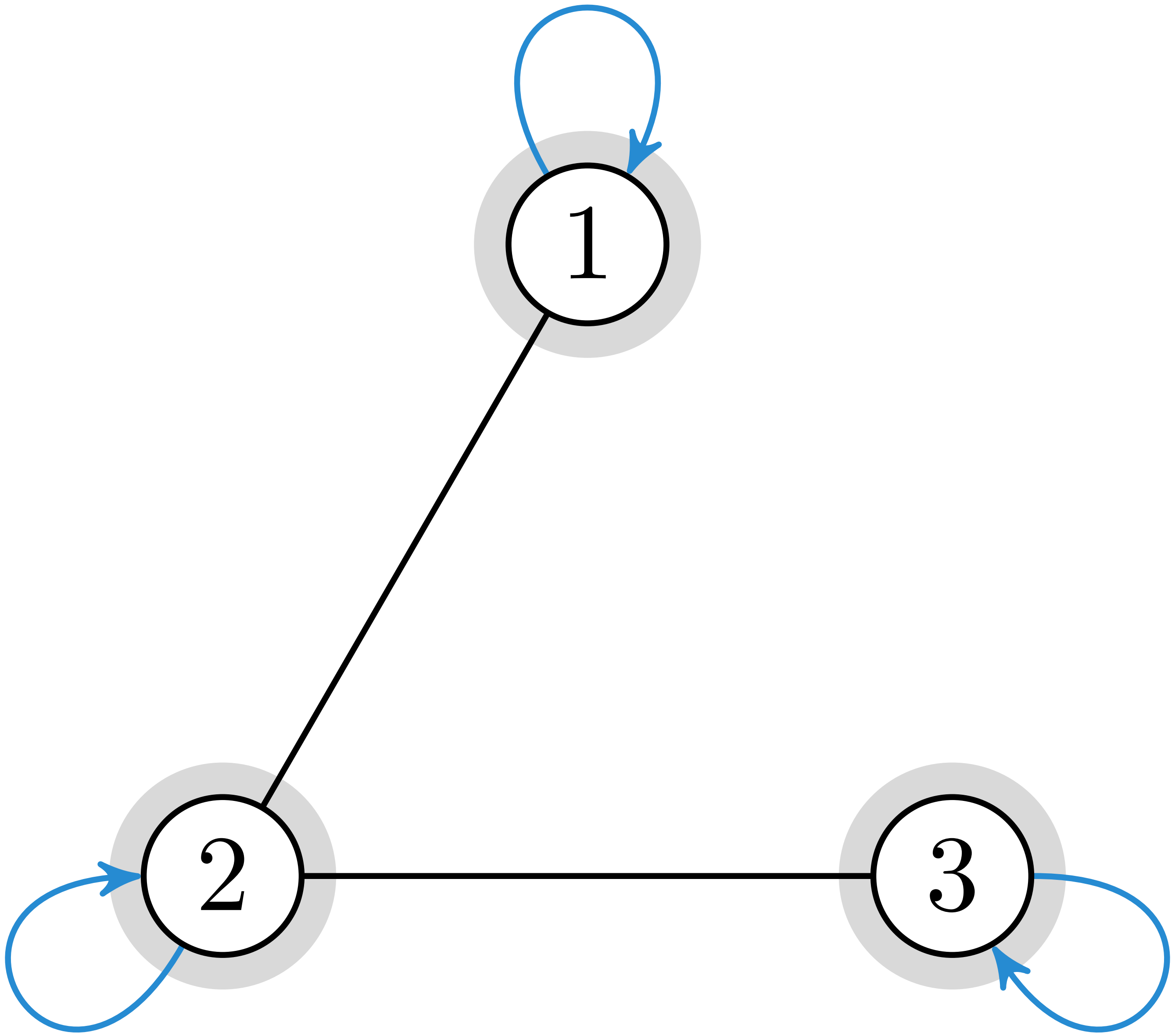
Han har også et par andre varianter i artikkelen – f.eks. å starte med hele grafen, og hele tiden fjerne den lengste kanten, så lenge resultatet er sammenhengende.

To skoger på én gang



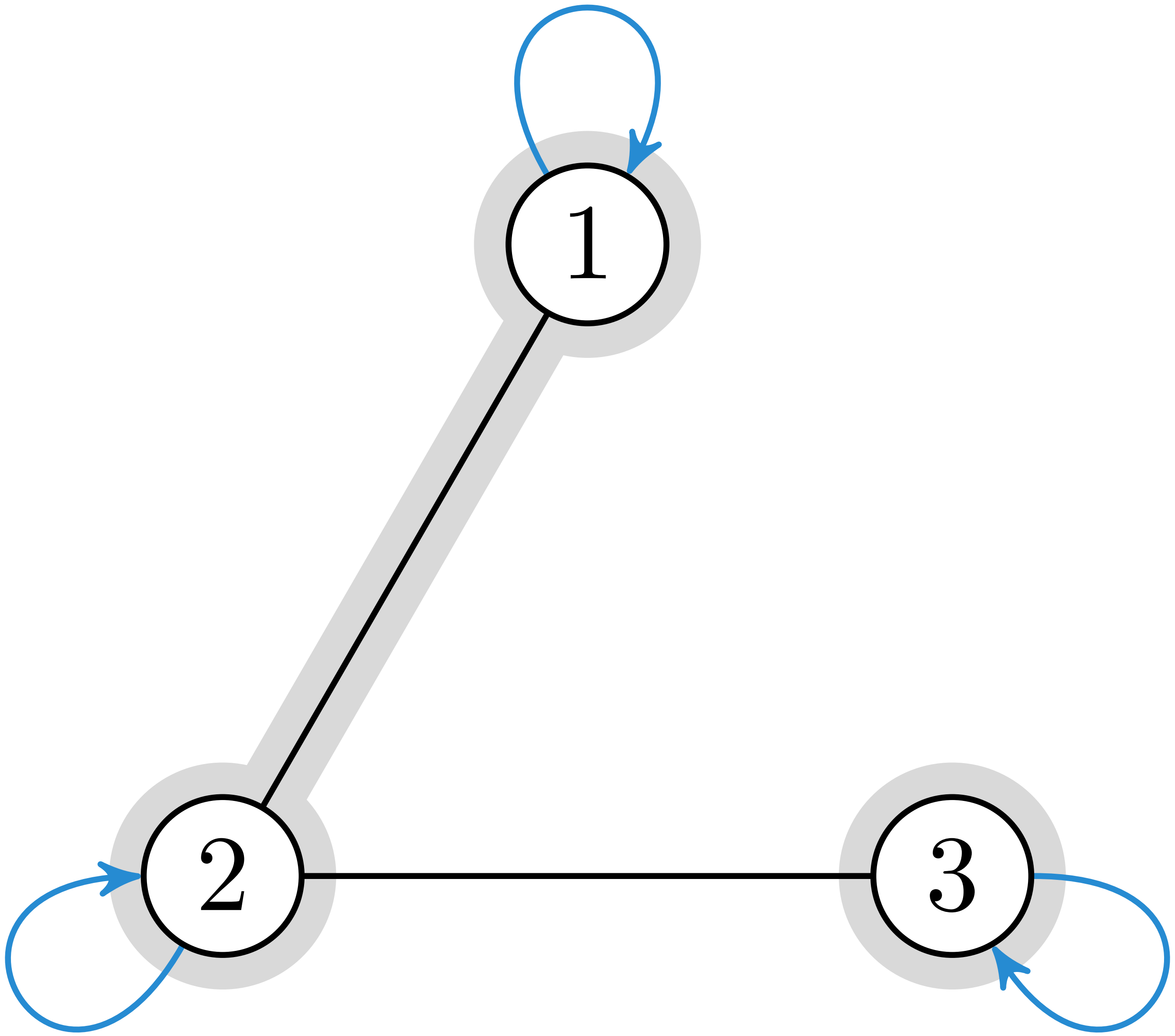
Til å begynne med er hver node en komponent i en partiell løsning

mlh@ntnu.no



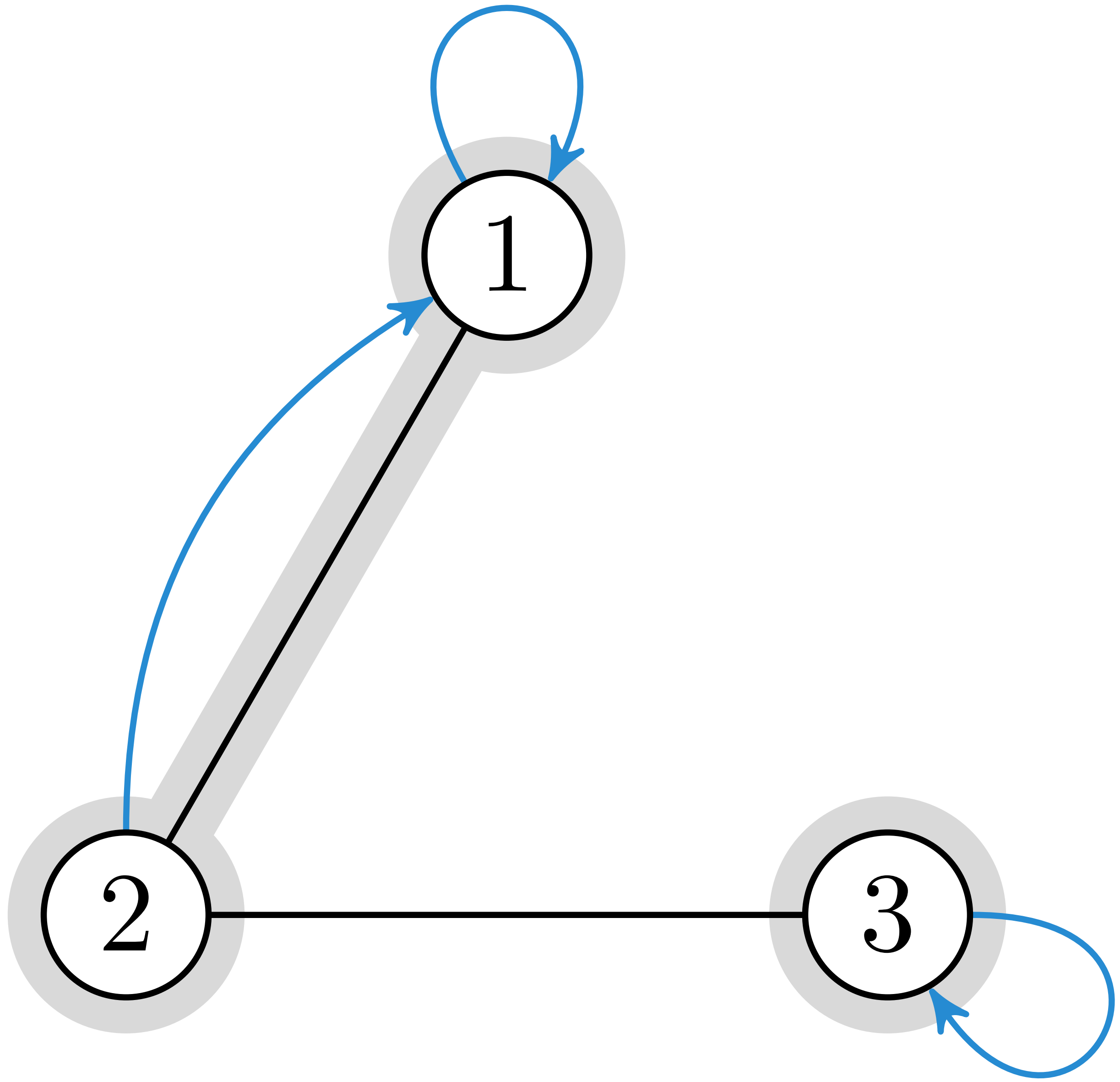
Komponenter: Disjunkte nodemengder. Starter med $v.p = v$

mlh@ntnu.no



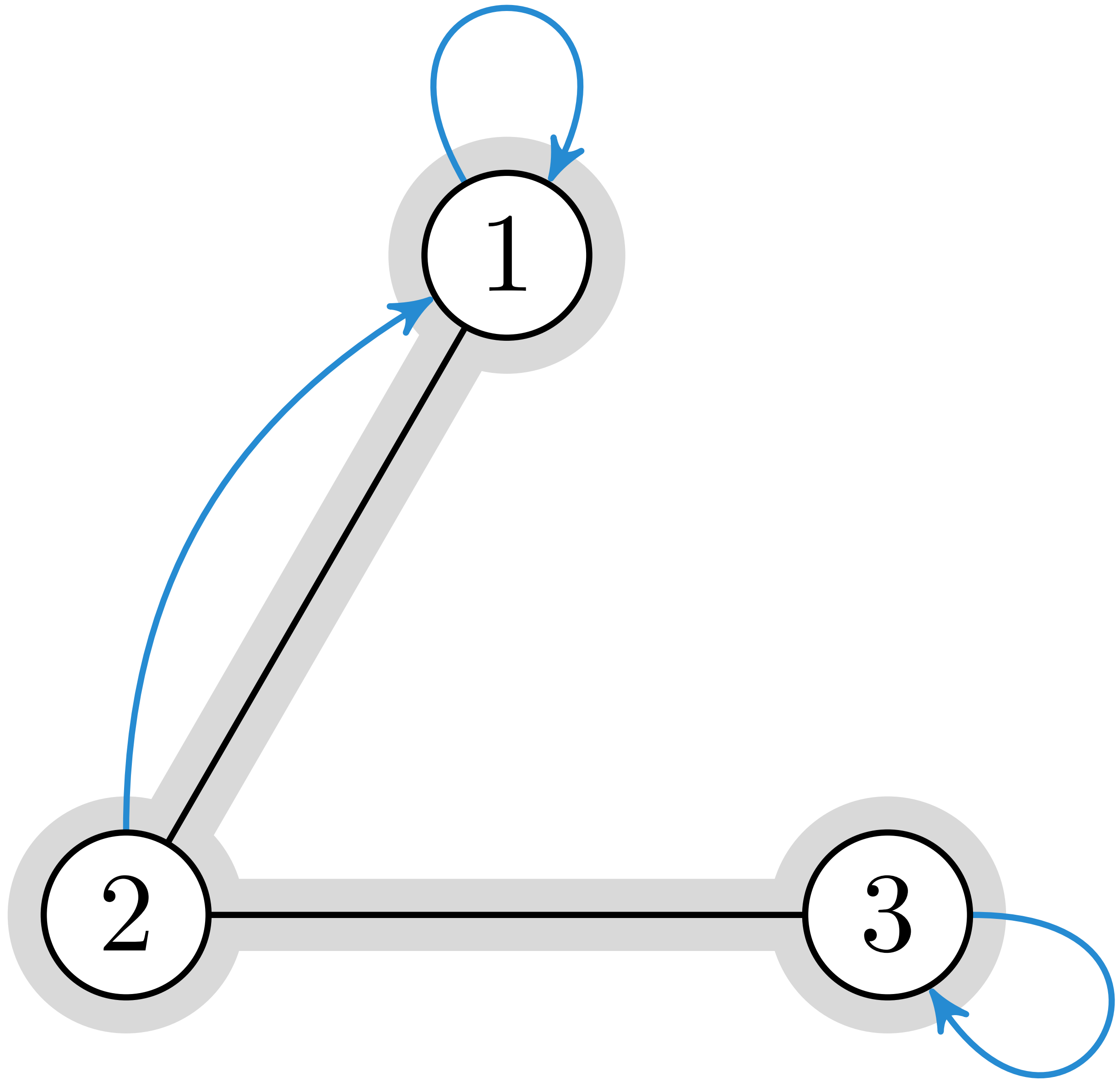
Hver kant kobler sammen to komponenter til et større tre

mlh@ntnu.no

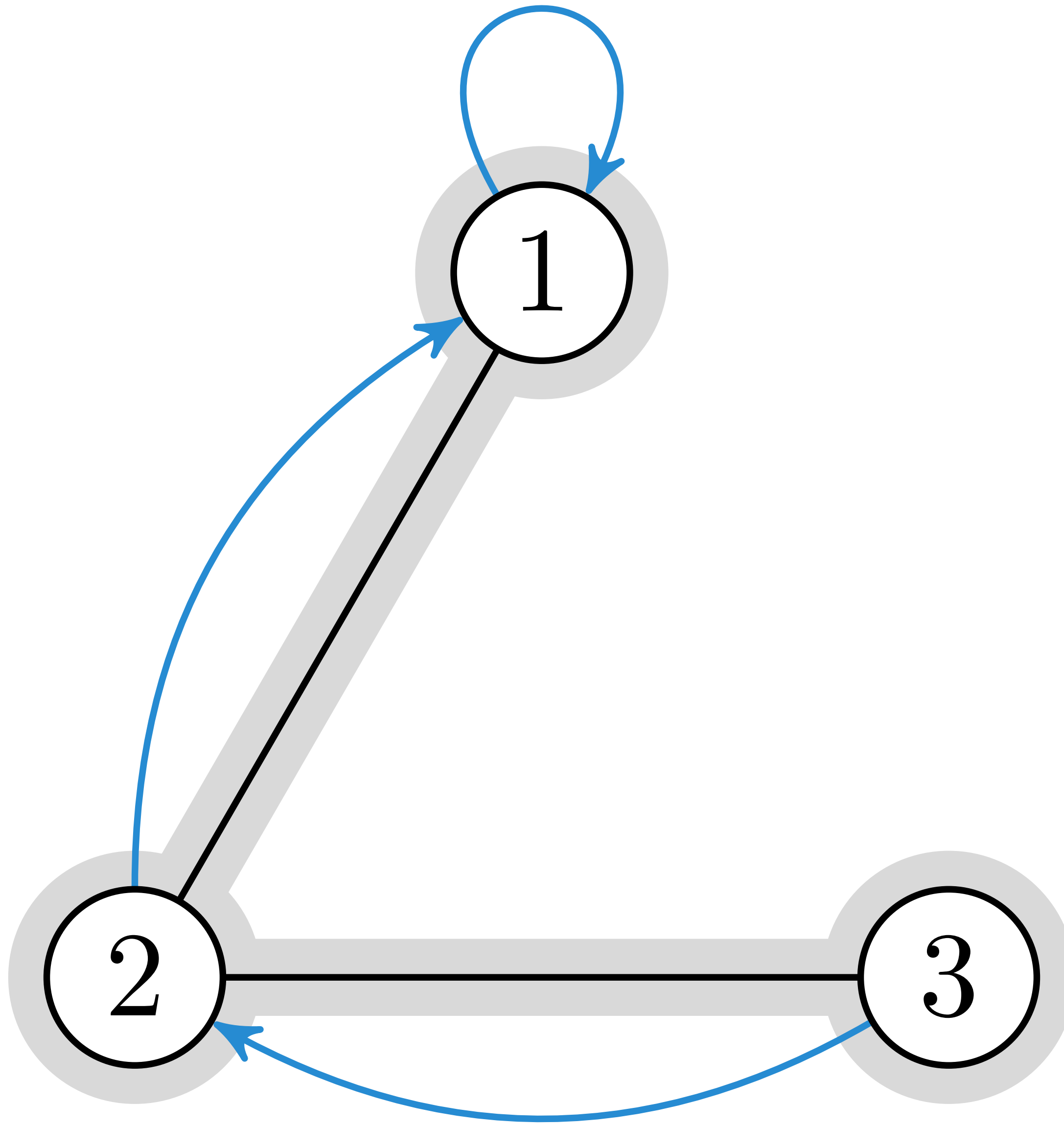


Vi kobler sammen komponentene ved å oppdatere $v.p$

mlh@ntnu.no

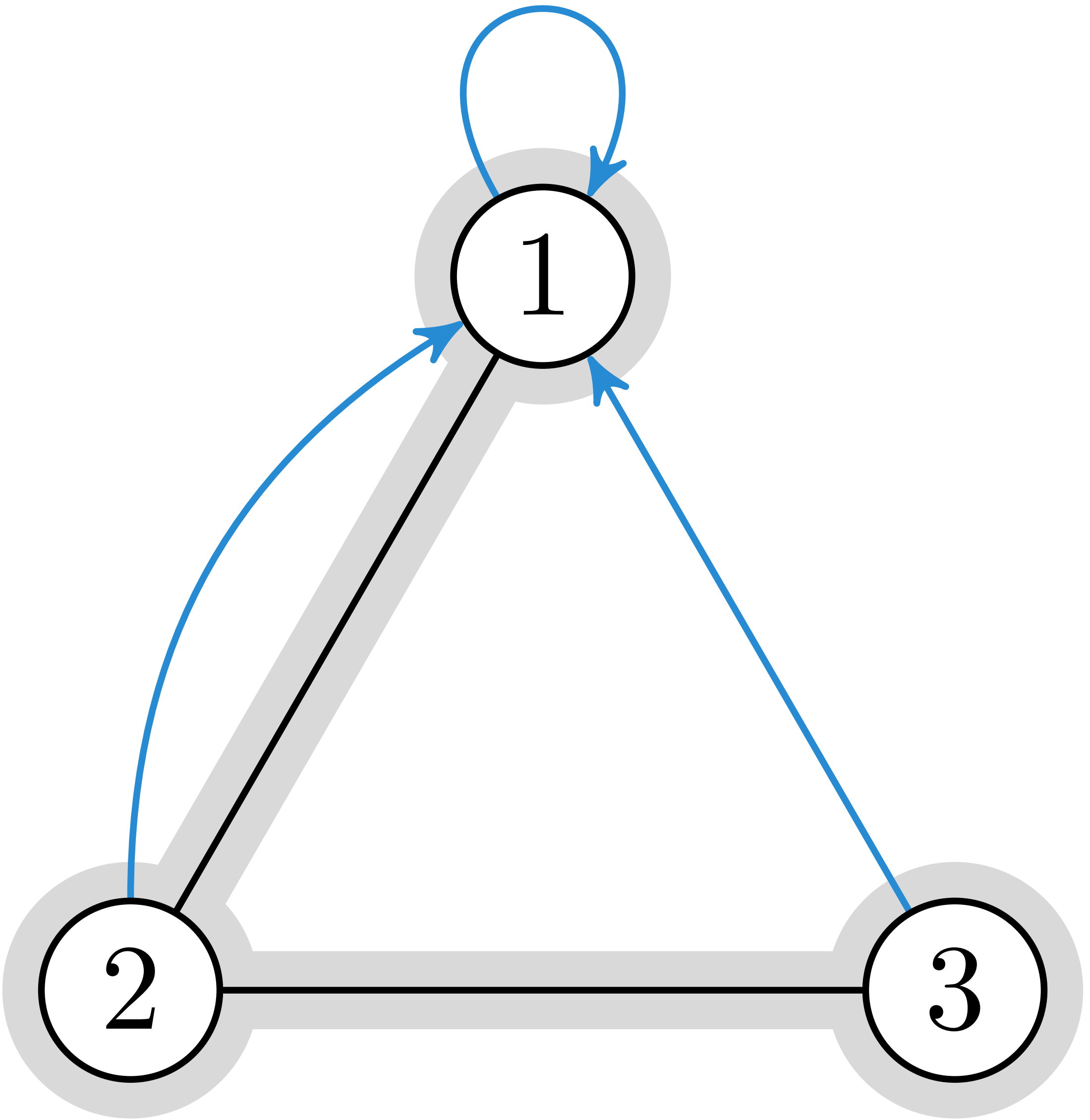


mlh@ntnu.no



Tilsammen så *kan* $v.p$-pekerne utgjøre MST-et vårt . . .

mlh@ntnu.no



Ofte mer effektivt om de ikke gjør det!

mlh@ntnu.no

- **Én skog er fragmenter av et MST**
- **Den andre skogen: Disjoint-set forest**
  - **Samme noder og komponenter**
  - **Rettede kanter/pekere som spiller en helt annen rolle**
- **Vi behandler denne siste skogen som en «black box» i algoritmen**

$\text{MST-K}\textsc{ruskal}(G, w)$

G graf
w vekting

Finn minimalt spenntre ved å velge minste lovlige kant

mlh@ntnu.no

MST-Kruskal(G, w)

1 A = ∅

G graf
w vekting
A kantutvalg

Kantene som skal utgjøre et minimalt spenntre til slutt

mlh@ntnu.no

MST-KRUSKAL(G, w)
1 $A = \emptyset$
2 **for** each vertex $v \in G.V$

G	graf
w	vekting
A	kantutvalg

Initialisering for mengdeoperasjonene våre

mlh@ntnu.no

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
```

G graf
w vekting
A kantutvalg

Vi starter med $|V|$ individuelle partielle spenntrær

mlh@ntnu.no

MST-KRUSKAL(G, w)
1 A = $\emptyset$
2 **for** each vertex $v \in$ G.V
3   MAKE-SET(v)
4 sort G.E by w

G graf
w vekting
A kantutvalg

Preprosessering: Vil hele tiden velge minste kant

mlh@ntnu.no

MST-KRUSKAL(G, w)
1 A = ∅
2 **for** each vertex $v \in$ G.V
3 MAKE-SET(v)
4 sort G.E by w
5 **for** each edge $(u, v) \in$ G.E

G graf
w vekting
A kantutvalg

Altså i stigende rekkefølge, etter vekt

mlh@ntnu.no

MST-KRUSKAL(G, w)

```
1  A = ∅
2  for each vertex v ∈ G.V
3          MAKE-SET(v)
4  sort G.E by w
5  for each edge (u, v) ∈ G.E
6          if FIND-SET(u) ≠ FIND-SET(v)
```

G graf
w vekting
A kantutvalg

Mellom to ulike komponenter av den partielle løsningen?

mlh@ntnu.no

MST-Kruskal(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    Make-Set(v)
sort $G.E$ by w
for each edge $(u, v) \in G.E$
    if Find-Set$(u) \neq$ Find-Set(v)
        $A = A \cup \{(u, v)\}$

G graf
w vekting
A kantutvalg

Da dannes ingen sykler, og vi kan velge kanten

mlh@ntnu.no

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort $G.E$ by w
for each edge $(u, v) \in G.E$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)

G graf
w vekting
A kantutvalg

Vi kobler sammen to partielle spenntrær til ett

mlh@ntnu.no

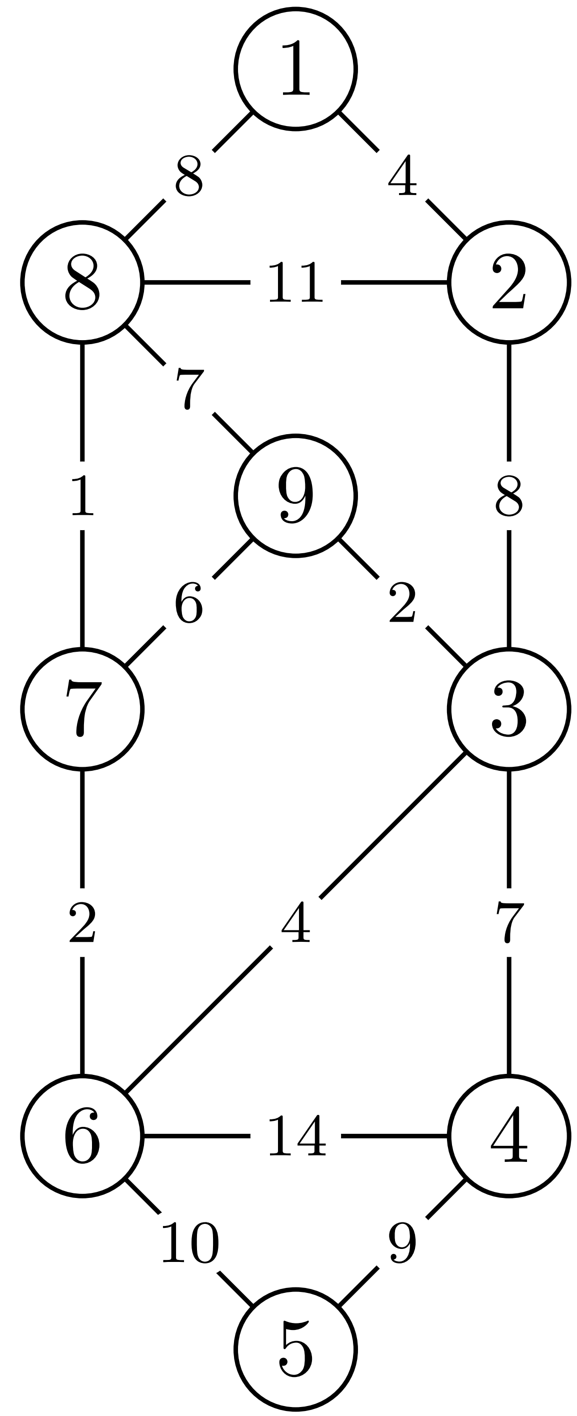
```
MST-Kruskal(G, w)
A = ∅
for each vertex v ∈ G.V
    Make-Set(v)
sort G.E by w
for each edge (u, v) ∈ G.E
    if Find-Set(u) ≠ Find-Set(v)
        A = A ∪ {(u, v)}
        Union(u, v)
return A
```

G	graf
w	vekting
A	kantutvalg

mlh@ntnu.no

MST-Kruskal(G, w)

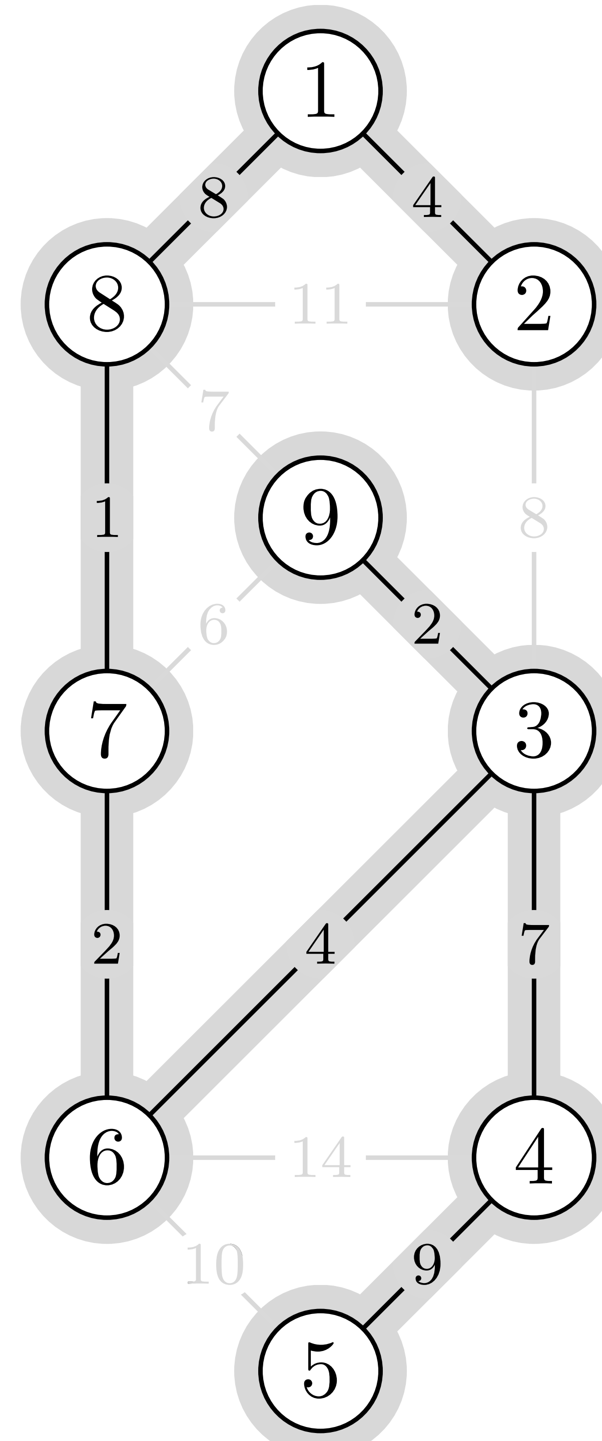
```
A = ∅
for each vertex v ∈ G.V
    Make-Set(v)
sort G.E by w
for each edge (u, v) ∈ G.E
    if Find-Set(u) ≠ Find-Set(v)
        A = A ∪ {(u, v)}
        Union(u, v)
return A
```



mlh@ntnu.no

MST-Kruskal(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
  Make-Set(v)
sort $G.E$ by w
for each edge $(u, v) \in G.E$
  if Find-Set$(u) \neq$ Find-Set(v)
    $A = A \cup \{(u, v)\}$
    Union(u, v)
return A



mlh@ntnu.no

Operasjon	Antall	Kjøretid

mlh@ntnu.no

Operasjon	Antall	Kjøretid
Make-Set	V	O(1)

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
MAKE-SET	V	O(1)

Merk: Boka kombinerer MAKE-SET med FIND-SET og UNION

mlh@ntnu.no

Operasjon	Antall	Kjøretid
MAKE-SET	V	$O(1)$
Sortering	1	$O(E \lg E)$

mlh@ntnu.no

Operasjon	Antall	Kjøretid
MAKE-SET	V	$O(1)$
Sortering	1	$O(E \lg E)$
FIND-SET	$O(E)$	$O(\alpha(V))$
UNION	$O(E)$	$O(\alpha(V))$

mlh@ntnu.no

Operasjon	Antall	Kjøretid
Make-Set	V	O(1)
Sortering	1	O(E lg E)
Find-Set	O(E)	O(α(V))
Union	O(E)	O(α(V))

$\alpha(n) = O(\lg n)$

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
Make-Set	V	$O(1)$
Sortering	1	$O(E \lg E)$
Find-Set	$O(E)$	$O(\alpha(V))$
Union	$O(E)$	$O(\alpha(V))$

$\alpha(n) = O(\lg n)$; i praksis har vi $\alpha(n) \leq 4$

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
MAKE-SET	V	O(1)
Sortering	1	O(E lg E)
FIND-SET	O(E)	O(lg V)
UNION	O(E)	O(lg V)

$\alpha(n) = \mathrm{O}(\lg n)$; i praksis har vi $\alpha(n) \leq 4$

mlh@ntnu.no

Operasjon	Antall	Kjøretid
MAKE-SET	V	O(1)
Sortering	1	O(E lg E)
FIND-SET	O(E)	O(lg V)
UNION	O(E)	O(lg V)

$|E| < |V|^2$

mlh@ntnu.no

Operasjon	Antall	Kjøretid
Make-Set	V	$O(1)$
Sortering	1	$O(E \lg E)$
Find-Set	$O(E)$	$O(\lg V)$
Union	$O(E)$	$O(\lg V)$

$$|E| < |V|^2 \implies \lg |E| < 2 \lg |V|$$

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
MAKE-SET	V	$O(1)$
Sortering	1	$O(E \lg E)$
FIND-SET	$O(E)$	$O(\lg V)$
UNION	$O(E)$	$O(\lg V)$

$$|E| < |V|^2 \implies \lg |E| < 2 \lg |V| \implies \lg E = O(\lg V)$$

mlh@ntnu.no

Operasjon	Antall	Kjøretid
MAKE-SET	V	O(1)
Sortering	1	O(E lg E)
FIND-SET	O(E)	O(lg V)
UNION	O(E)	O(lg V)

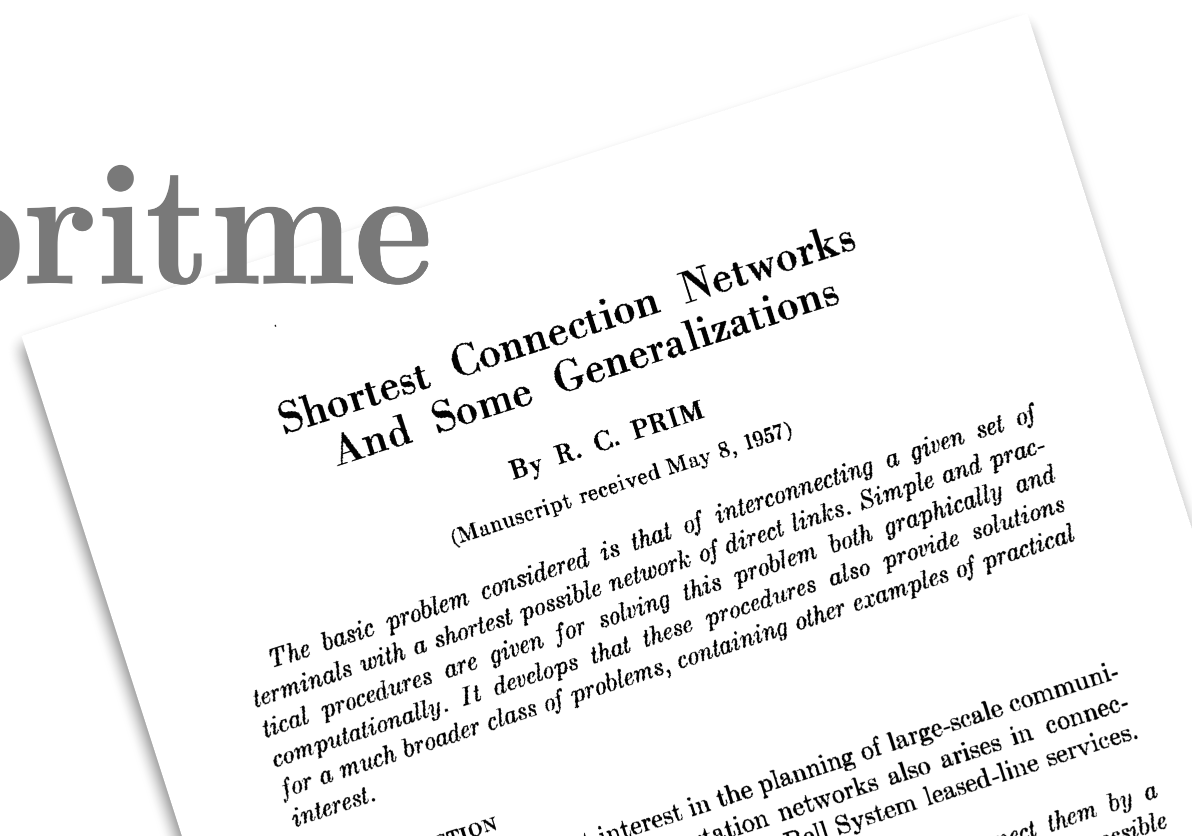
Totalt: $O(E \lg V)$

$$|E| < |V|^2 \implies \lg |E| < 2 \lg |V| \implies \lg E = O(\lg V)$$

mlh@ntnu.no

4:4

Prims algoritme


Shortest Connection Networks And Some Generalizations

By R. C. PRIM

(Manuscript received May 8, 1957)

The basic problem considered is that of interconnecting a given set of terminals with a shortest possible network of direct links. Simple and practical procedures are given for solving this problem both graphically and computationally. It develops that these procedures also provide solutions for a much broader class of problems, containing other examples of practical interest.

... interest in the planning of large-scale communi- ... tation networks also arises in connec- ... Bell System leased-line services.

- **Kan implementeres vha. traversering**
- **Der BFS bruker FIFO og DFS bruker LIFO, så bruker Prim en min-prioritets-kø**
- **Prioriteten er vekten på den letteste kanten mellom noden og treet**
- **For enkelhets skyld: Legg alle noder inn fra starten, med uendelig dårlig prioritet**

$\text{MST-P}{\scriptsize\text{RIM}}(\mathrm{G}, w, r)$

G graf
w vekting
r startnode

Start i r. Gjenta: Inkludér noden nærmest treet

mlh@ntnu.no

MST-PRIM(G, w, r)
1 **for** each $u \in G.V$

G graf
w vekting
r startnode

Initialisering

mlh@ntnu.no

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
```

G graf
w vekting
r startnode

Vekt til letteste kant til treet

mlh@ntnu.no

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$

G graf
w vekting
r startnode

Noden den letteste kanten kommer fra; utgjør MST til slutt

mlh@ntnu.no

MST-PRIM(G, w, r)

```
for each u ∈ G.V
        u.key = ∞
        u.π = NIL
r.key = 0
```

G graf
w vekting
r startnode

r er startnoden vår

mlh@ntnu.no

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$

G graf
w vekting
r startnode
Q pri-kø

Vi kunne ha traversert … men trenger ikke

mlh@ntnu.no

MST-PRIM(G, w, r)

```
for each u ∈ G.V
        u.key = ∞
        u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
```

G graf
w vekting
r startnode
Q pri-kø

Som ved traversering: Står det noe på huskelista vår?

mlh@ntnu.no

MST-Prim(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \textsc{nil}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \textsc{Extract-Min}(Q)$

G	graf
w	vekting
r	startnode
Q	pri-kø

Noden nærmest treet

mlh@ntnu.no

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{Extract-Min}(Q)$
    for each $v \in G.Adj[u]$

G	graf
w	vekting
r	startnode
Q	pri-kø

«Oppdag» – eller i hvert fall oppdatér – naboene

mlh@ntnu.no

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{Extract-Min}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$

G graf
w vekting
r startnode
Q pri-kø

Er v fortsatt ubrukt? Fant vi en bedre kant fra treet til v?

mlh@ntnu.no

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
```

G graf
w vekting
r startnode
Q pri-kø

Da bruker vi denne nye kanten i stedet . . .

mlh@ntnu.no

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$

G graf
w vekting
r startnode
Q pri-kø

…og oppdaterer prioriteten til v i Q

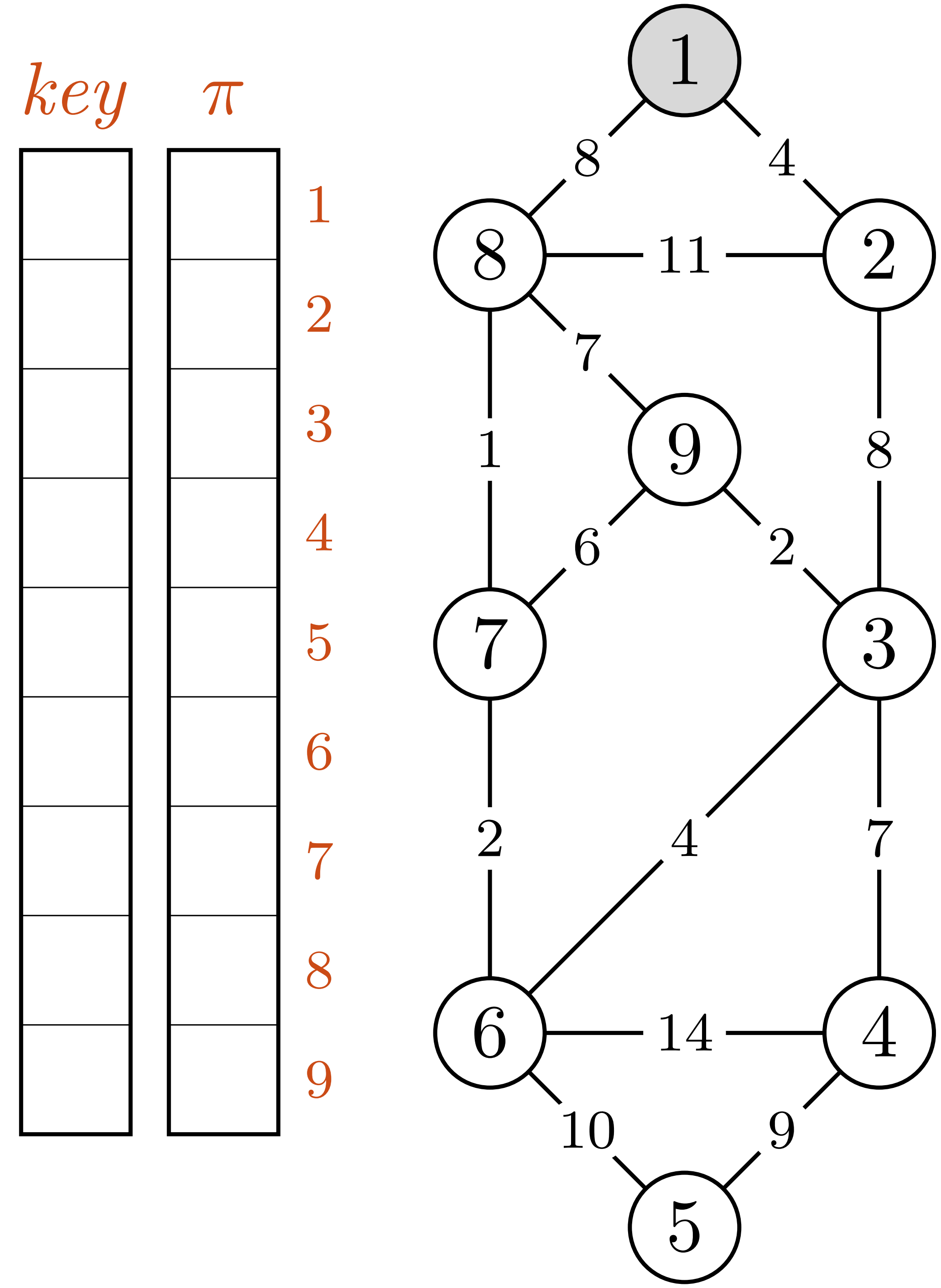
mlh@ntnu.no

- **I det følgende: Farging som for BFS**
- **Kanter mellom svarte noder er endelige**
- **Beste kanter for grå noder også uthevet**
- **Boka uthever bare kantene i spenntreet**

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

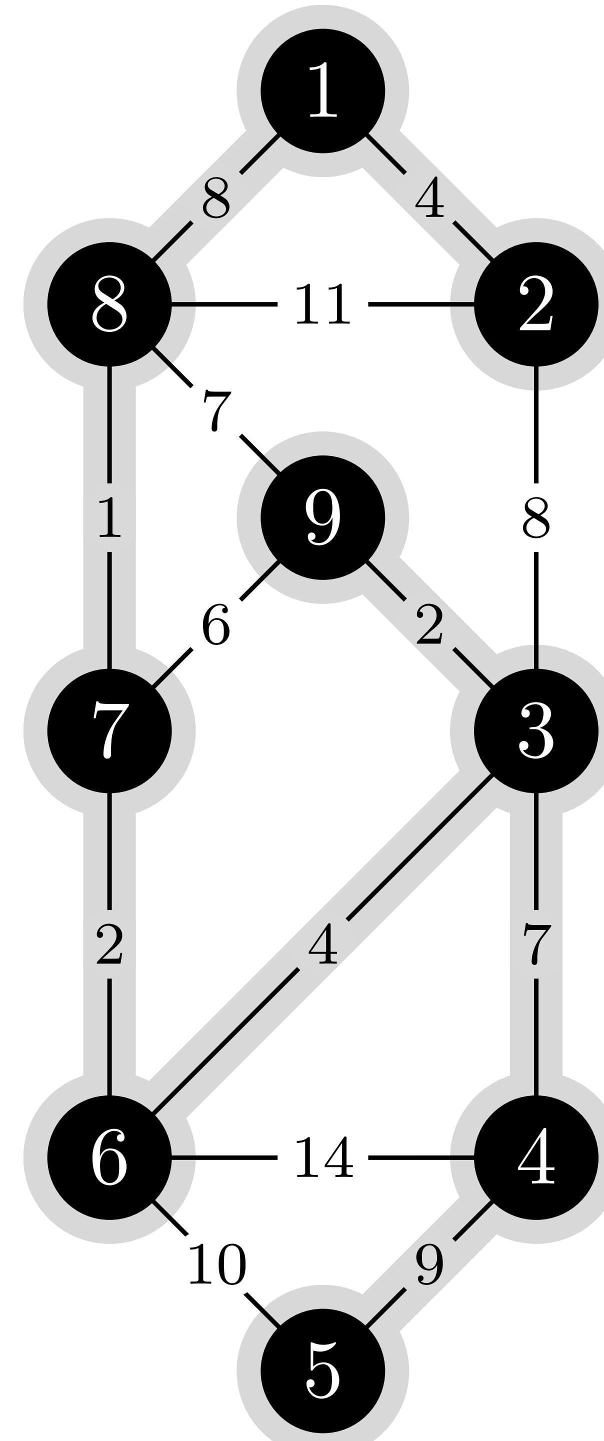
$u, v \ = \ -, -$



mlh@ntnu.no

```
MST-Prim(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

key	π	
0	–	1
4	1	2
4	6	3
7	3	4
9	4	5
2	7	6
1	8	7
8	1	8
2	3	9



$u, v \;=\; 5, \text{-}$

mlh@ntnu.no

Operasjon	Antall	Kjøretid

mlh@ntnu.no

Operasjon	Antall	Kjøretid
Build-Min-Heap	1	O(V)

Vi legger alle noder i køen til å begynne med

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
BUILD-MIN-HEAP	1	$O(V)$
EXTRACT-MIN	V	$O(\lg V)$

Alle noder må hentes ut av køen én gang, før de besøkes

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
Build-Min-Heap	1	O(V)
Extract-Min	V	O(lg V)
Decrease-Key	E	O(lg V)

Langs hver kant (u, v) vi finner, oppdatér prioriteten til v

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
Build-Min-Heap	1	$O(V)$
Extract-Min	V	$O(\lg V)$
Decrease-Key	E	$O(\lg V)$

Totalt: $O(E \lg V)$

mlh@ntnu.no

Operasjon	Antall	Kjøretid
BUILD-MIN-HEAP	1	$O(V)$
EXTRACT-MIN	V	$O(\lg V)$
DECREASE-KEY	E	$O(\lg V)$

Totalt: $O(E \lg V)$

Dette gjelder om vi bruker en binærhaug

mlh@ntnu.no

Operasjon	Antall	Kjøretid
Build-Min-Heap	1	$O(V)$
Extract-Min	V	$O(\lg V)$
Decrease-Key	E	$O(\lg V)$

Totalt: $O(E \lg V)$

$O(1)$ amortisert for Fib.-haug

Dette gjelder om vi bruker en binærhaug

mlh@ntnu.no

Operasjon	**Antall**	**Kjøretid**
BUILD-MIN-HEAP	1	$O(V)$
EXTRACT-MIN	V	$O(\lg V)$
DECREASE-KEY	E	$O(\lg V)$

Totalt: $O(E \lg V)$

$O(1)$ amortisert for Fib.-haug

Kan forbedres til $O(E + V \lg V)$ med Fibonacci-haug

mlh@ntnu.no

1. Disjunkte mengder

2. Generisk MST

3. Kruskals algoritme

4. Prims algoritme