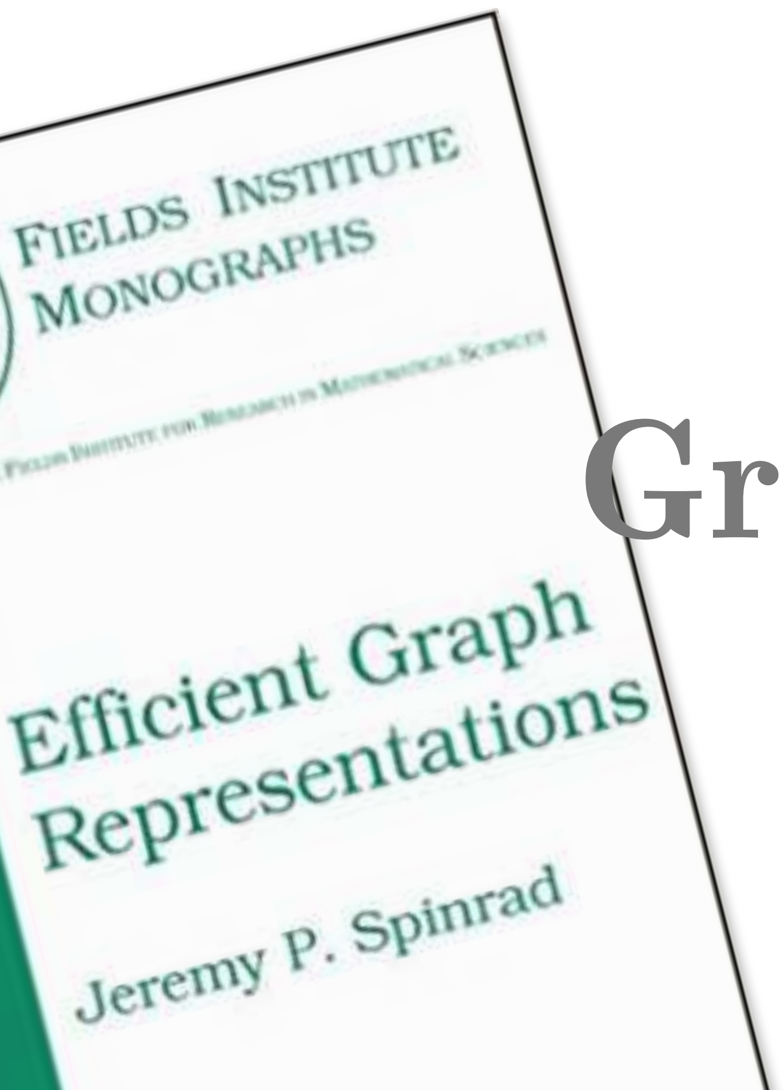
Forelesning 8

Traversering av grafer

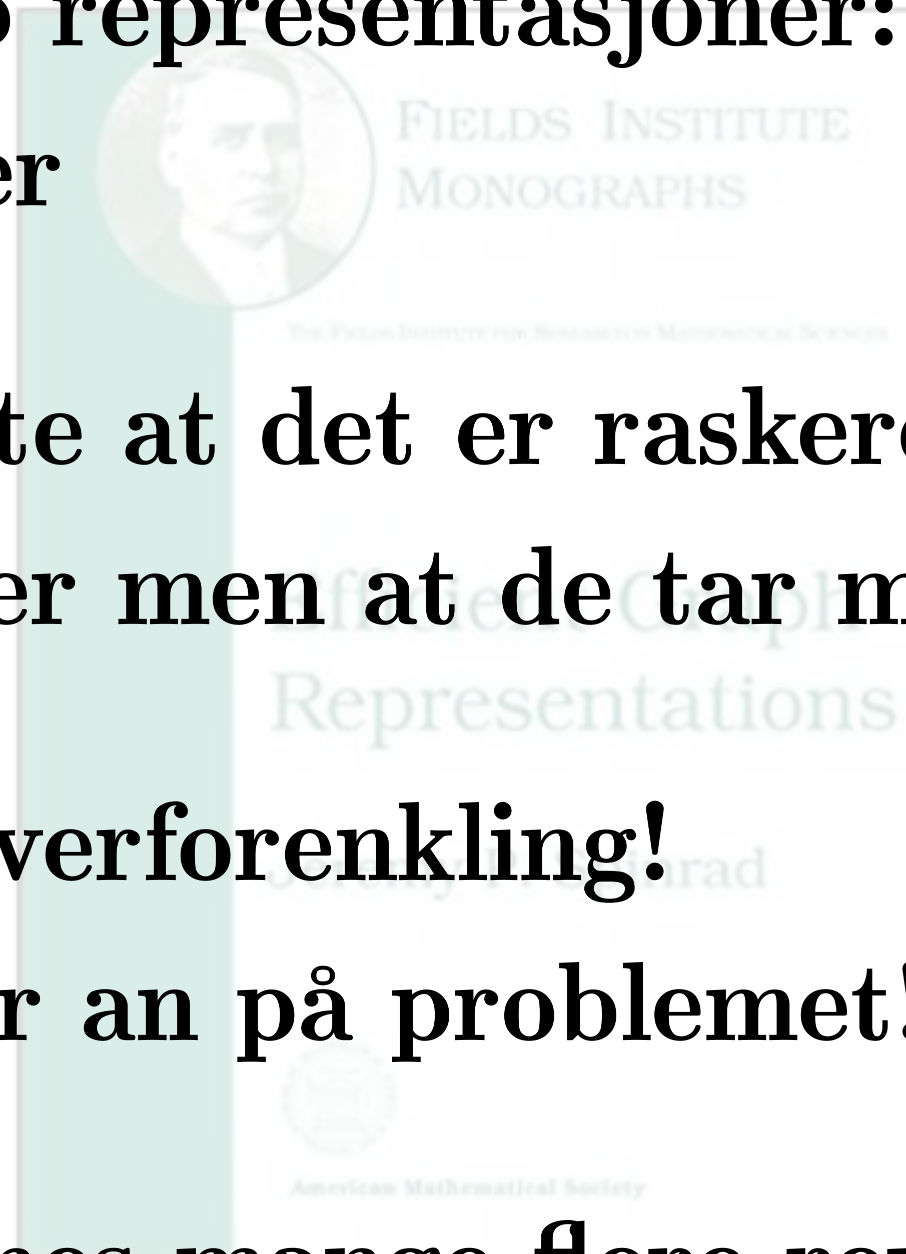
1. **Grafrepresentasjoner**
2. **Bredde-først-søk**
3. **Dybde-først-søk**
4. **Topologisk sortering**

1:4

Grafrepresentasjoner



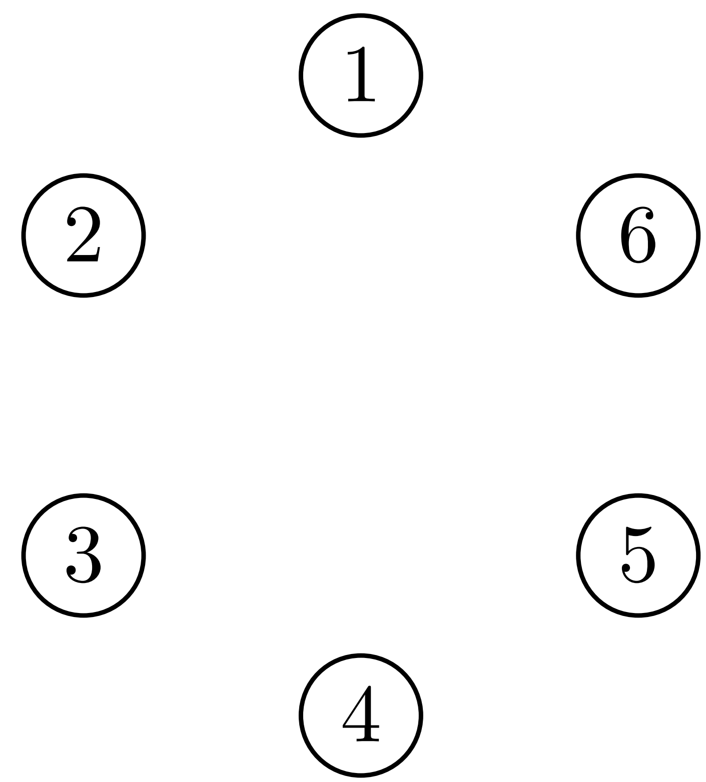
- **Vi ser på to representasjoner: Nabomatriser og nabolister**
- **Man sier ofte at det er raskere med nabomatriser men at de tar mer plass**
- **Det er en overforenkling!**
  Det kommer an på problemet!
- **Og: Det finnes mange flere representasjoner**



Se «Efficient Graph Representations» av Jeremy P. Spinrad for en mer detaljert diskusjon.

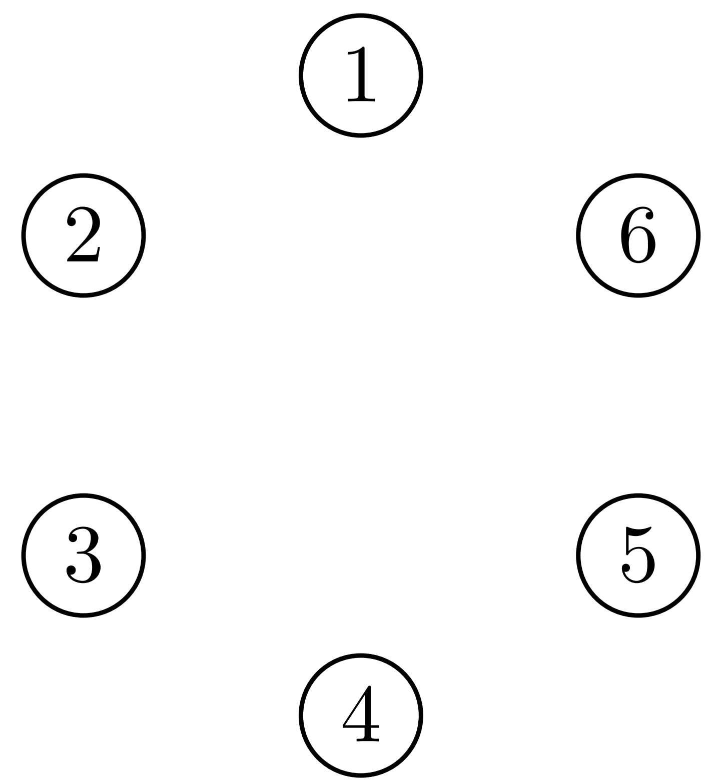
Grafrepresentasjoner › Nabomatriser

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0



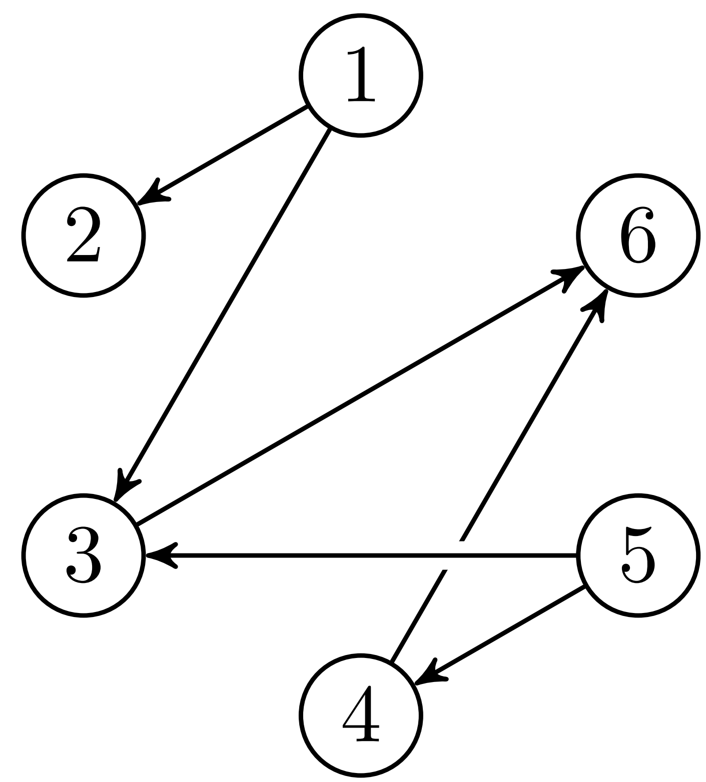
Nabomatrise: $\mathrm{A}[u, v] \iff (u, v) \in \mathrm{G.E}$

	1	2	3	4	5	6
1		0	0			
2						
3						0
4						0
5			0	0		
6						



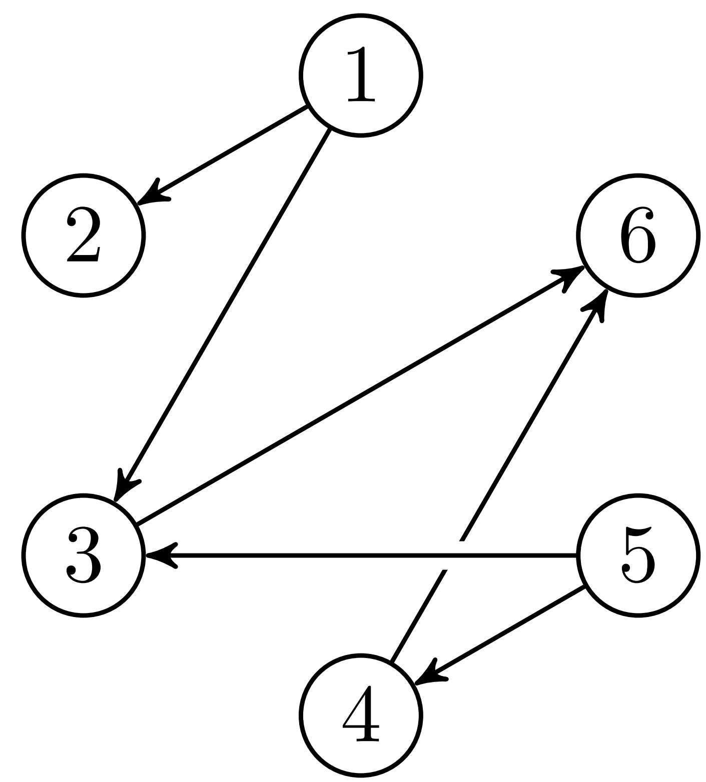
Nabomatrise: $\mathrm{A}[u,v] \iff (u,v) \in \mathrm{G.E}$

	1	2	3	4	5	6
1		1	1			
2						
3						1
4						1
5			1	1		
6						



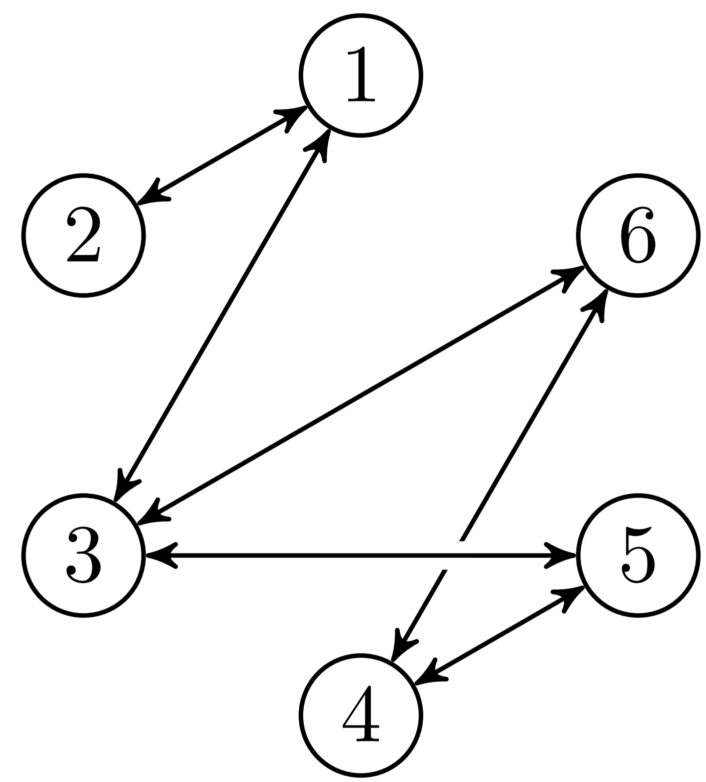
Nabomatrise: $\quad \mathrm{A}[u, v] \iff (u, v) \in \mathrm{G.E}$

	1	2	3	4	5	6
1		1	1			
2						
3						1
4						1
5			1	1		
6						



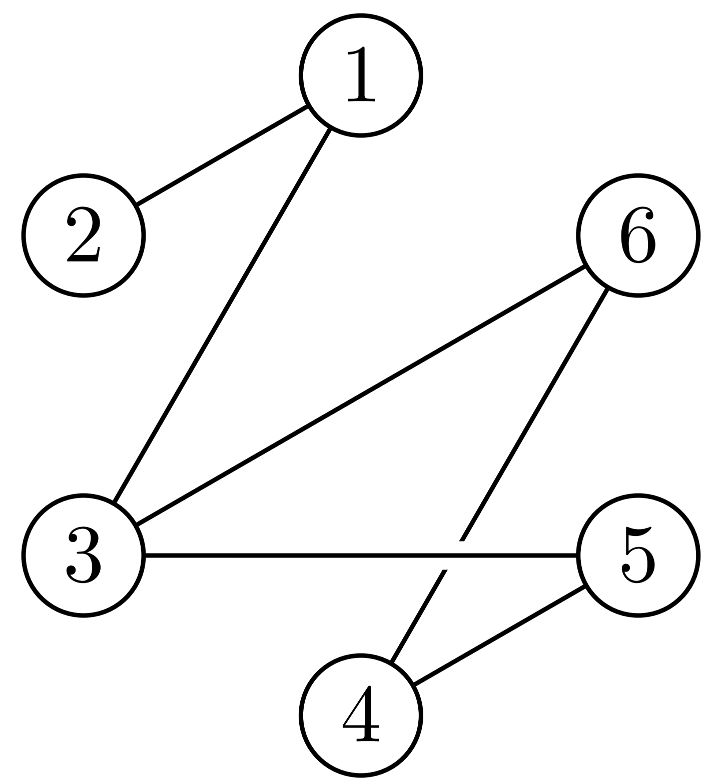
Urettet G $\iff$ symmetrisk A: $\mathrm{A}[u, v] = \mathrm{A}[v, u]$

	1	2	3	4	5	6
1		1	1			
2	1					
3	1				1	1
4					1	1
5			1	1		
6			1	1		



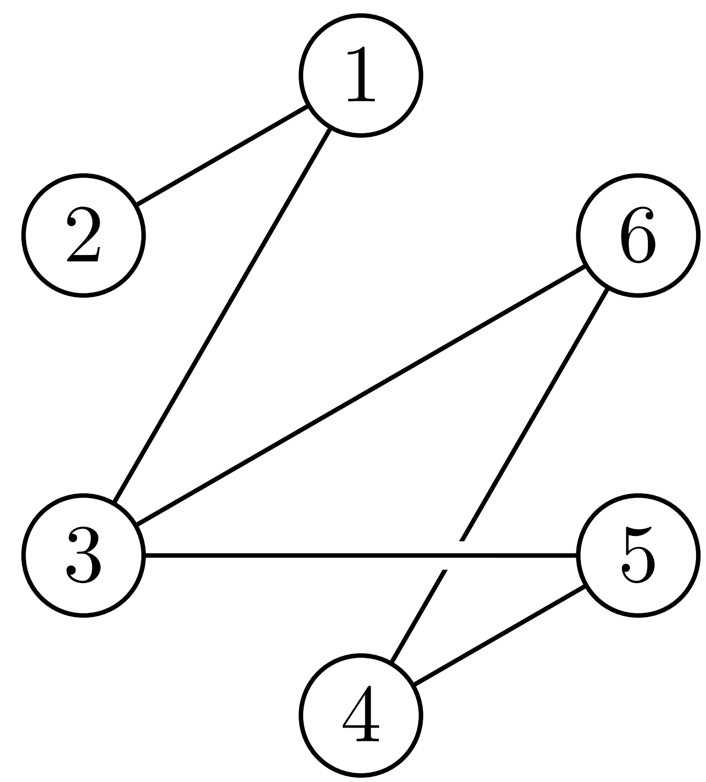
Urettet G $\iff$ symmetrisk A: $\mathrm{A}[u, v] = \mathrm{A}[v, u]$

	1	2	3	4	5	6
1		1	1			
2	1					
3	1				1	1
4					1	1
5			1	1		
6			1	1		



Urettet G $\iff$ symmetrisk A: $\mathrm{A}[u,v] = \mathrm{A}[v,u]$

	1	2	3	4	5	6
1		1	1			
2	1					
3	1				1	1
4					1	1
5			1	1		
6			1	1		

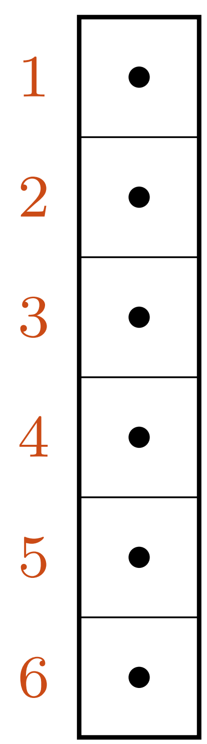


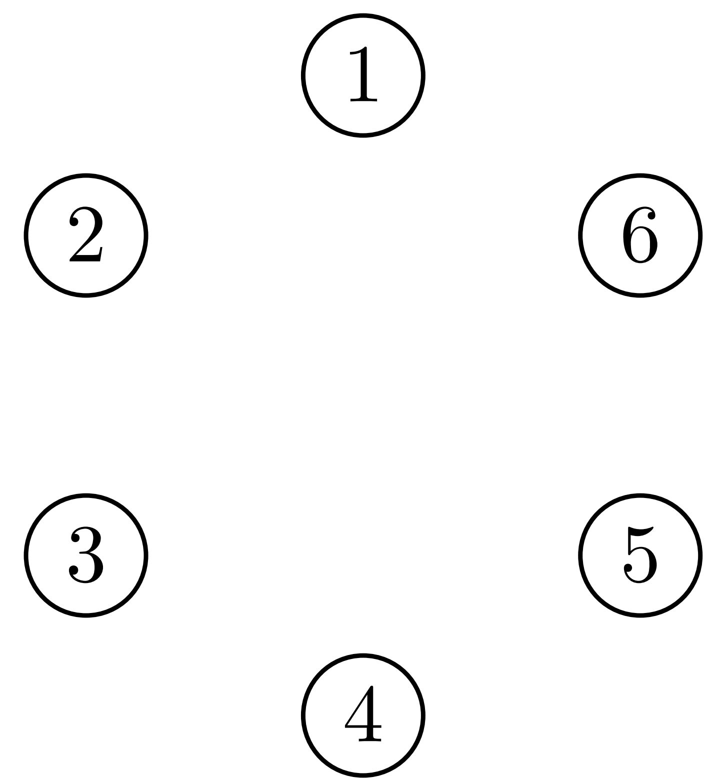
Egnet til raske oppslag; ikke så egnet til traversering

Grafrepresentasjoner ›

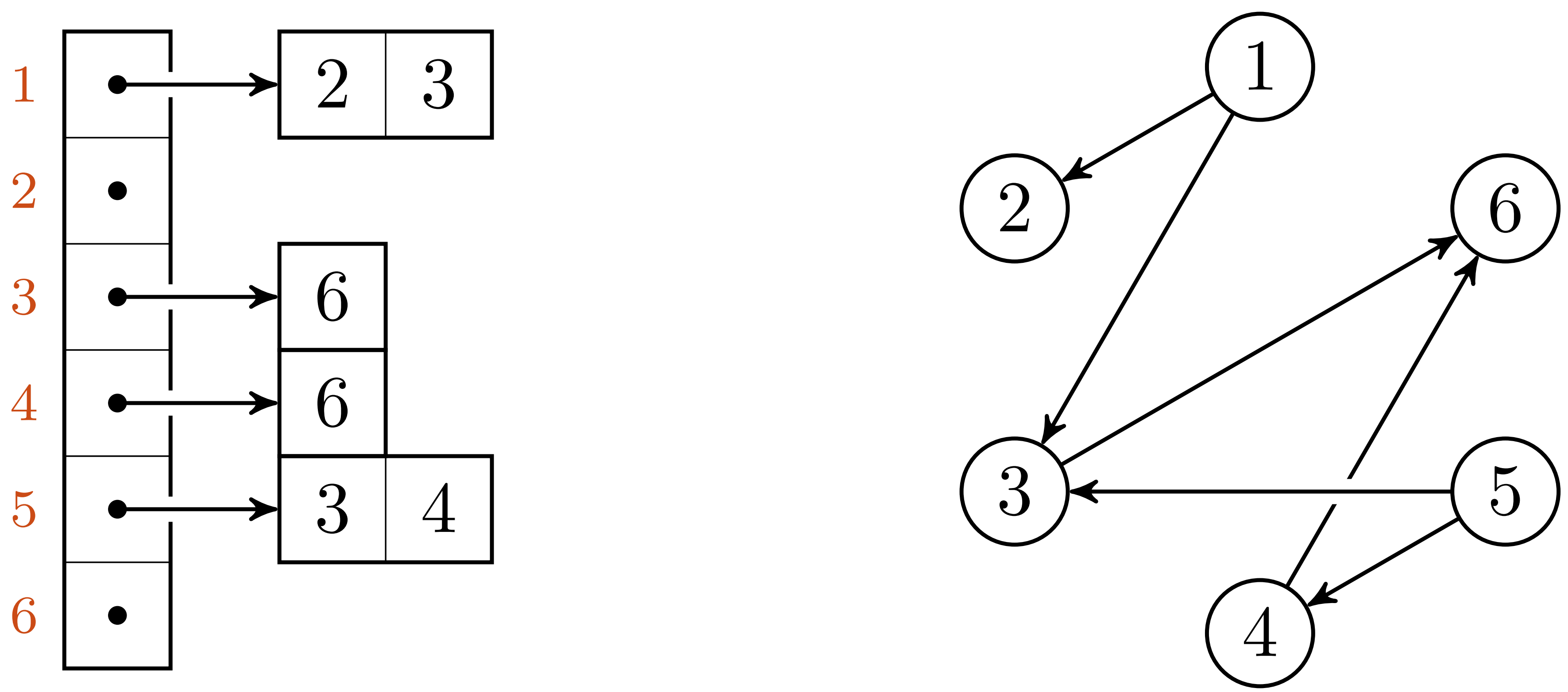
Nabolister

Det er altså snakk om én liste per node – ikke én liste totalt. Dvs., grafen representeres (i naboliste-representasjonen) av en tabell med mange nabolister – *ikke* «en naboliste».

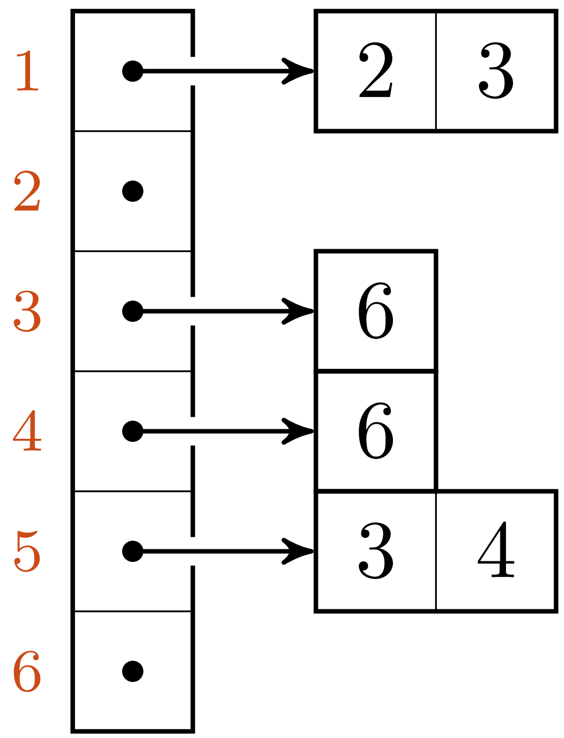


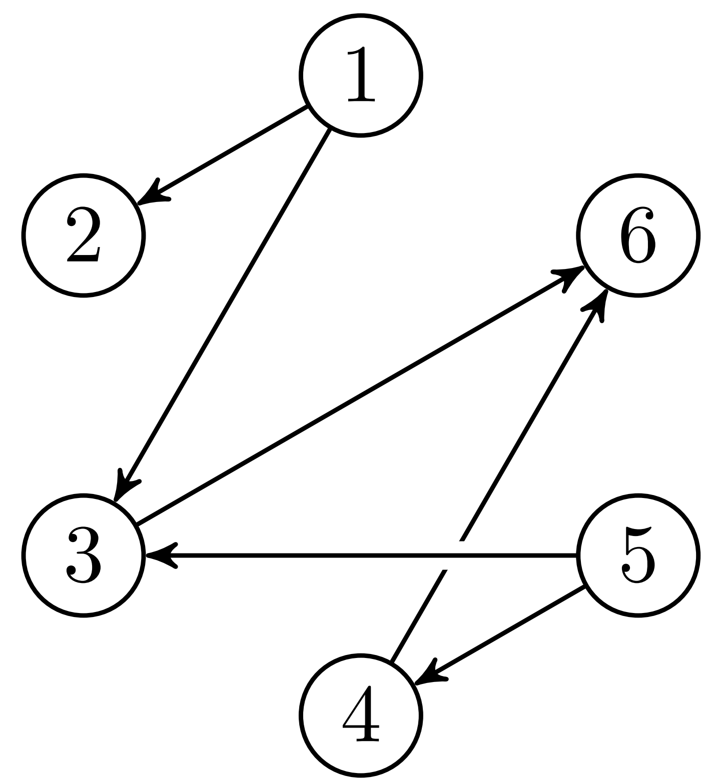


Nabolister: Liste (eller tabell) med ut-naboer for hver node

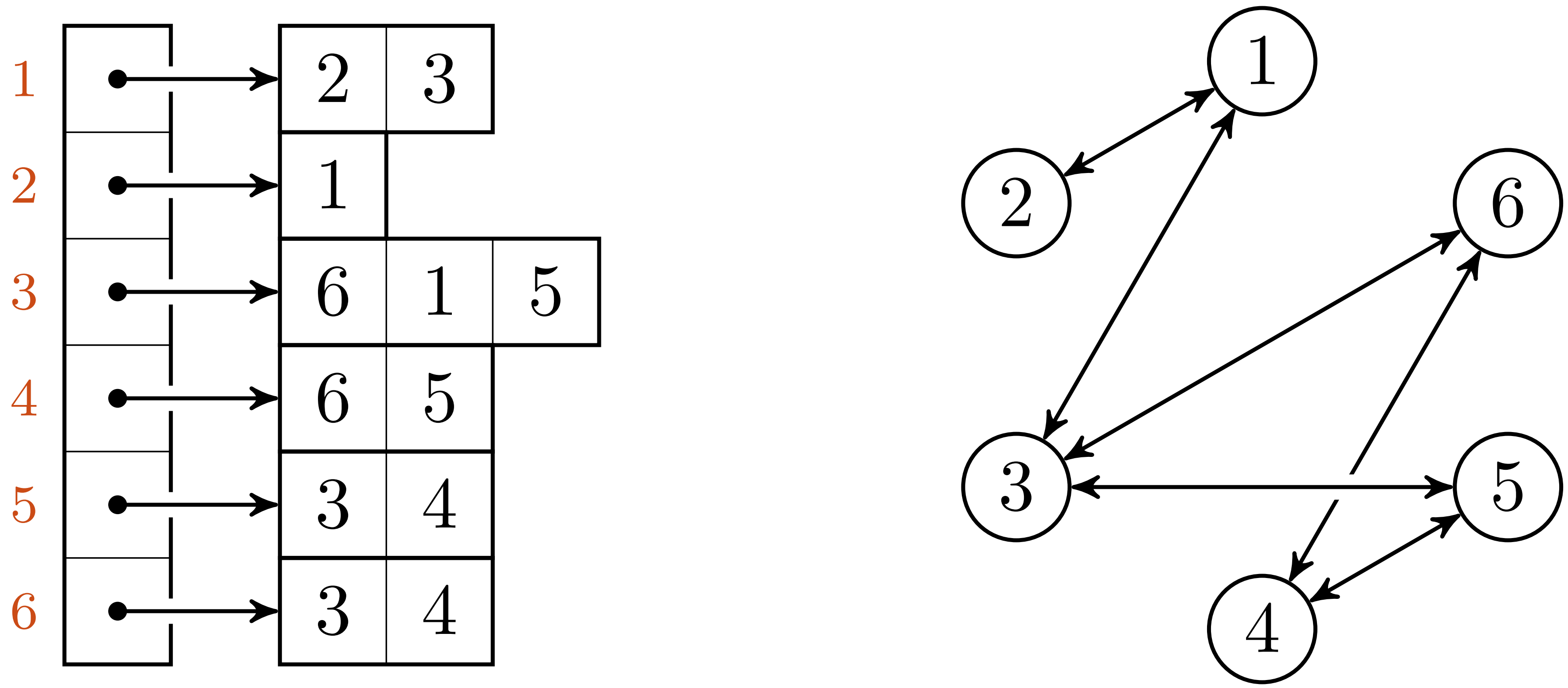


Nabolister: Liste (eller tabell) med ut-naboer for hver node

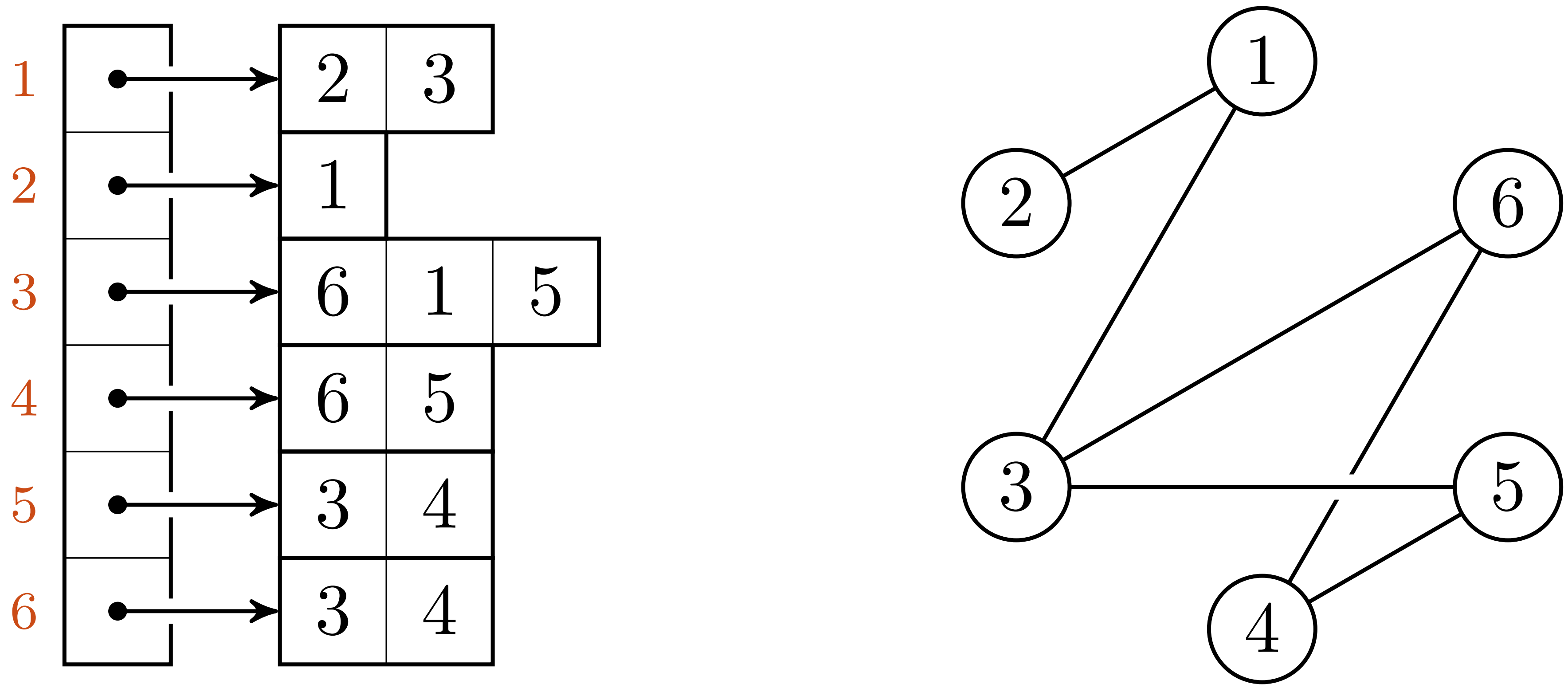




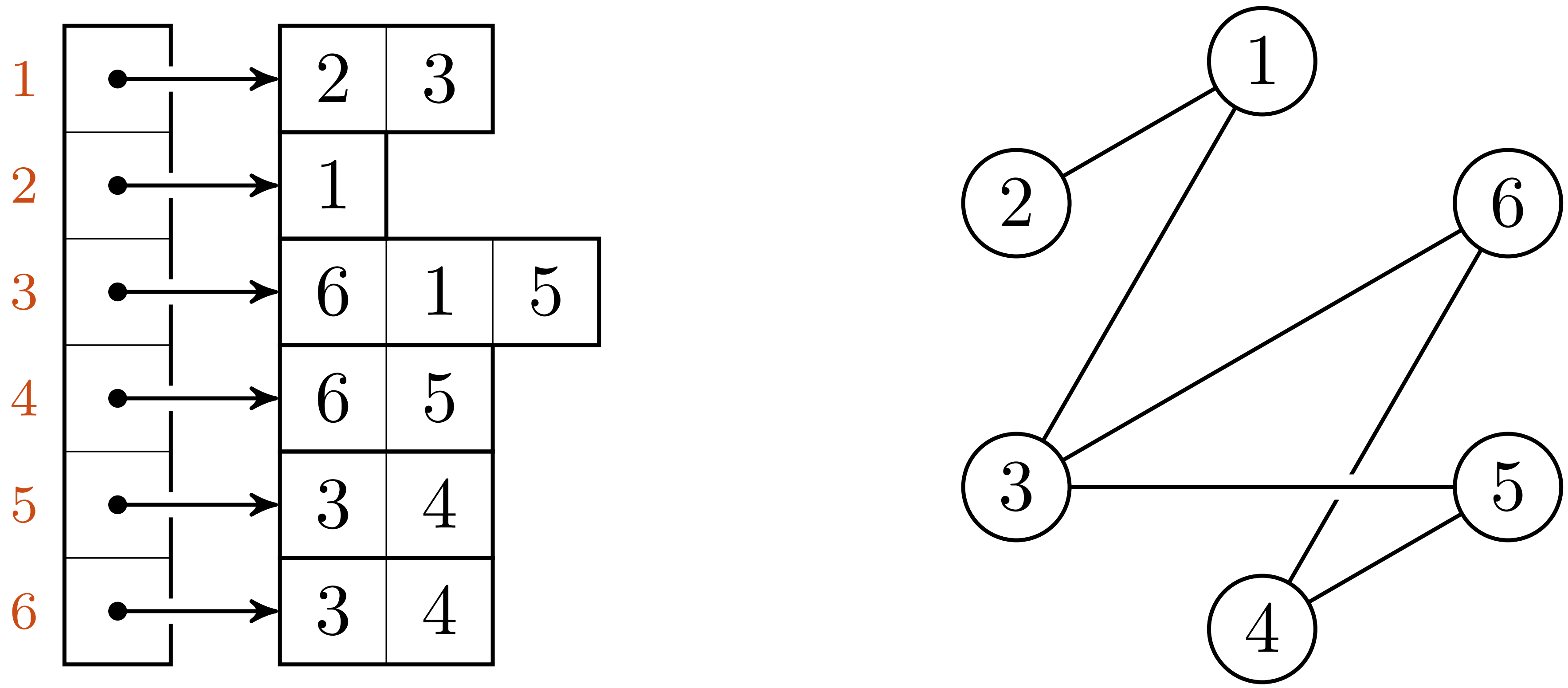
Urettet G $\iff$ kanter går begge veier



Urettet G $\iff$ kanter går begge veier



Urettet G $\iff$ kanter går begge veier



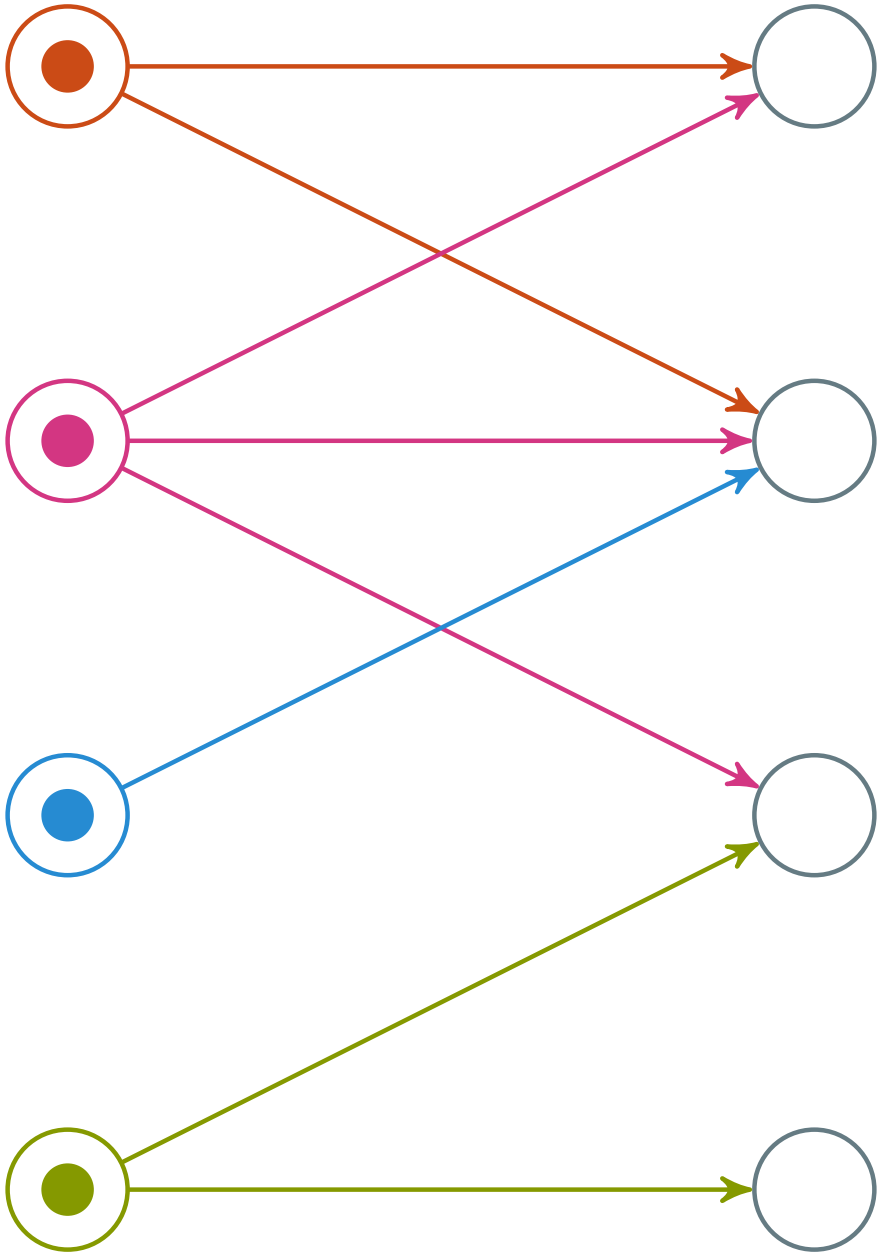
Kompakt; egnet til traversering; ikke så egnet til raske oppslag

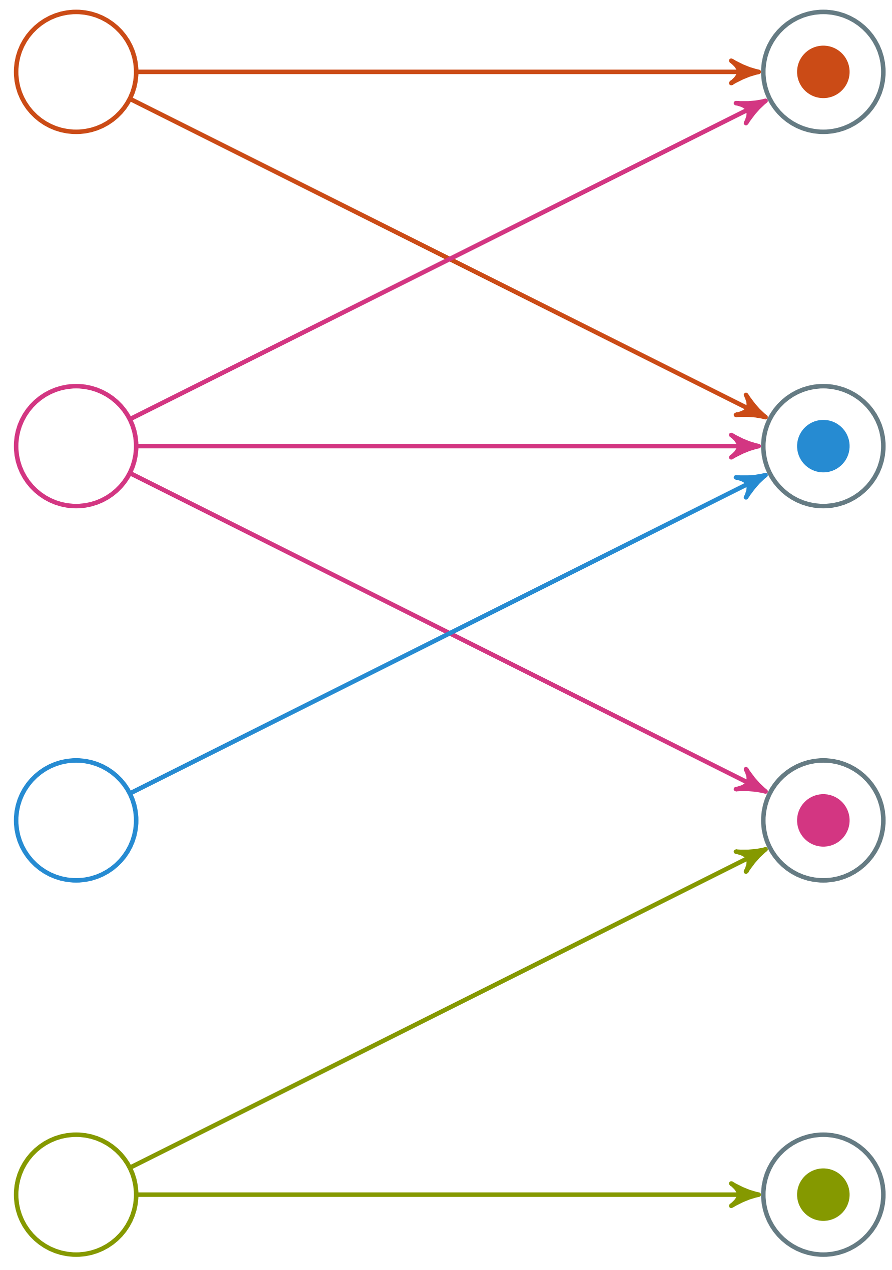
Kan godt «blande inn» hashtabeller e.l.

Nabomatriser egner seg til direkte oppslag. Nabolister egner seg til traversering. Nabolister tar også mindre plass dersom grafen har få kanter – men ikke ellers!

Traversering

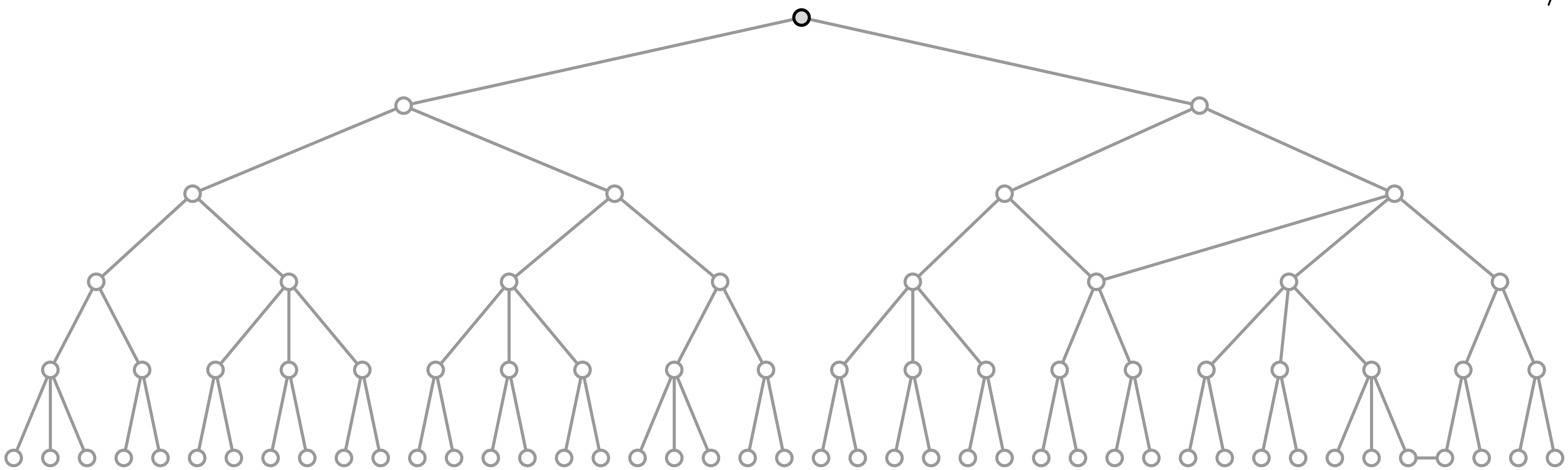
Matching som motivasjon

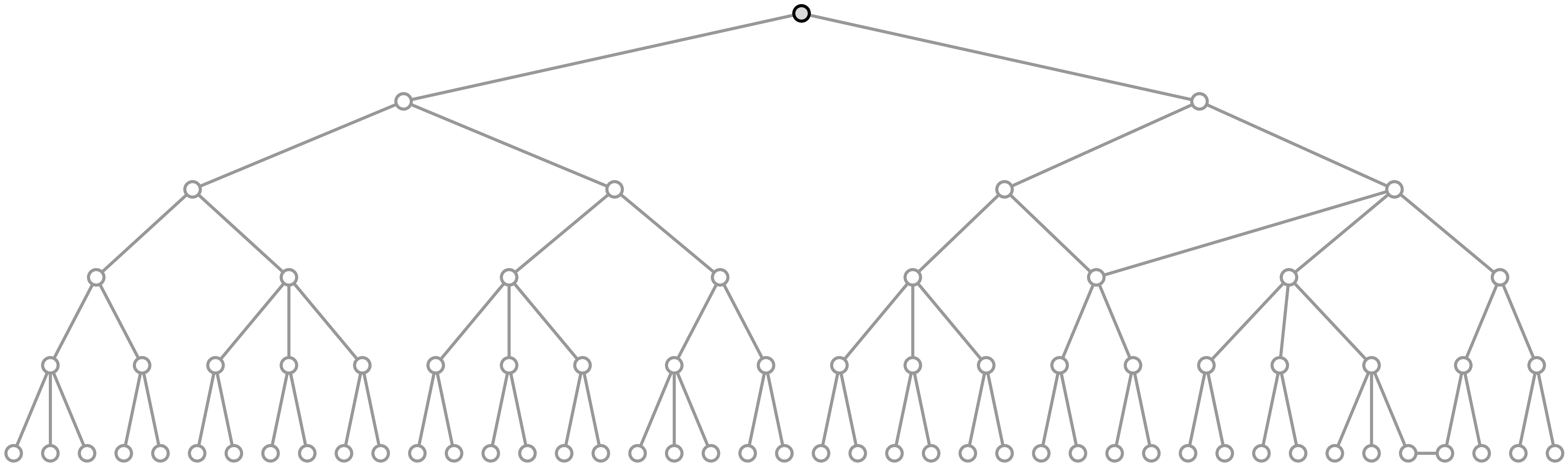




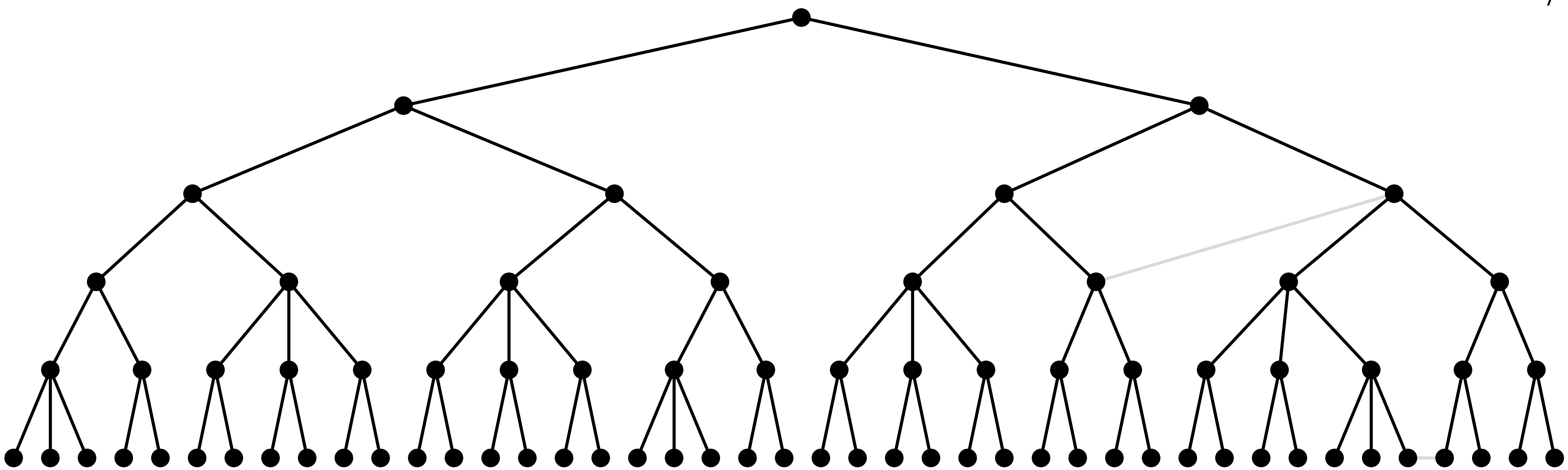
Slike stier finner vi vha. traversering

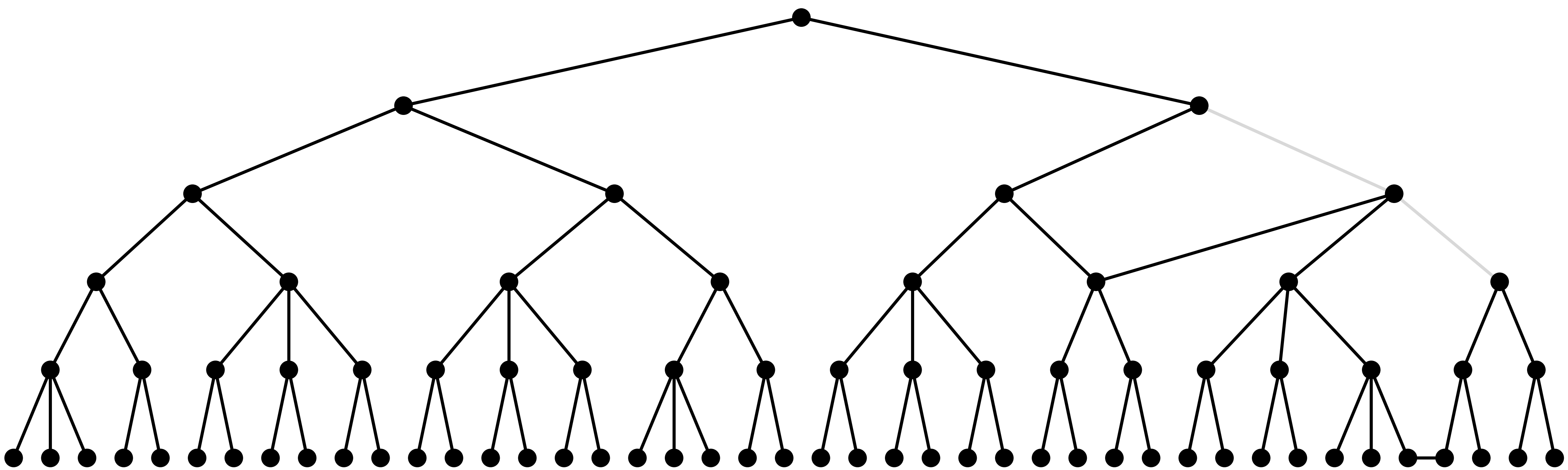
Vi kommer tilbake til matcheproblemet i forelesning 12 – men løsningen krever traversering, så la oss se på det!





mlh@ntnu.no



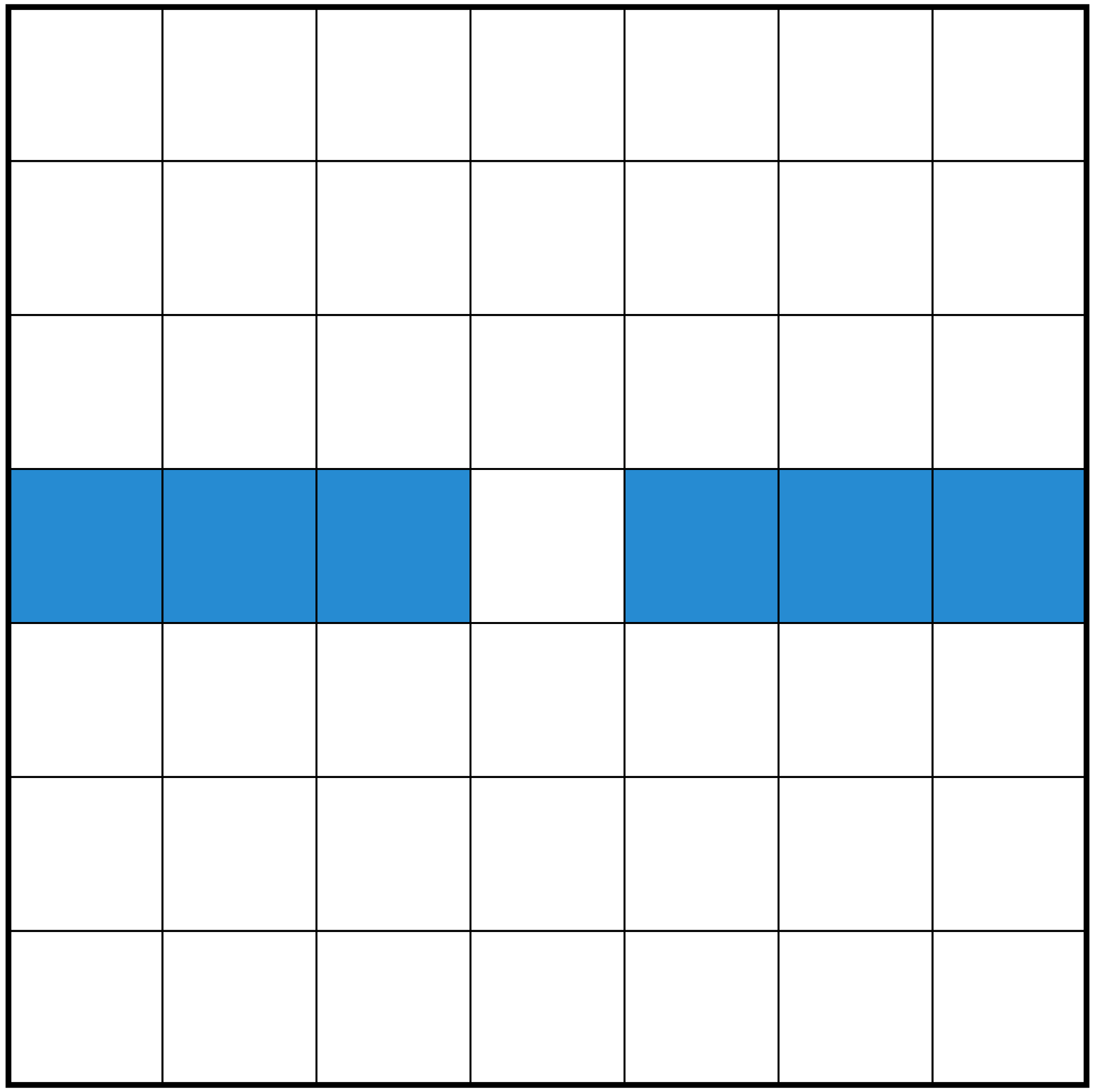


2:4

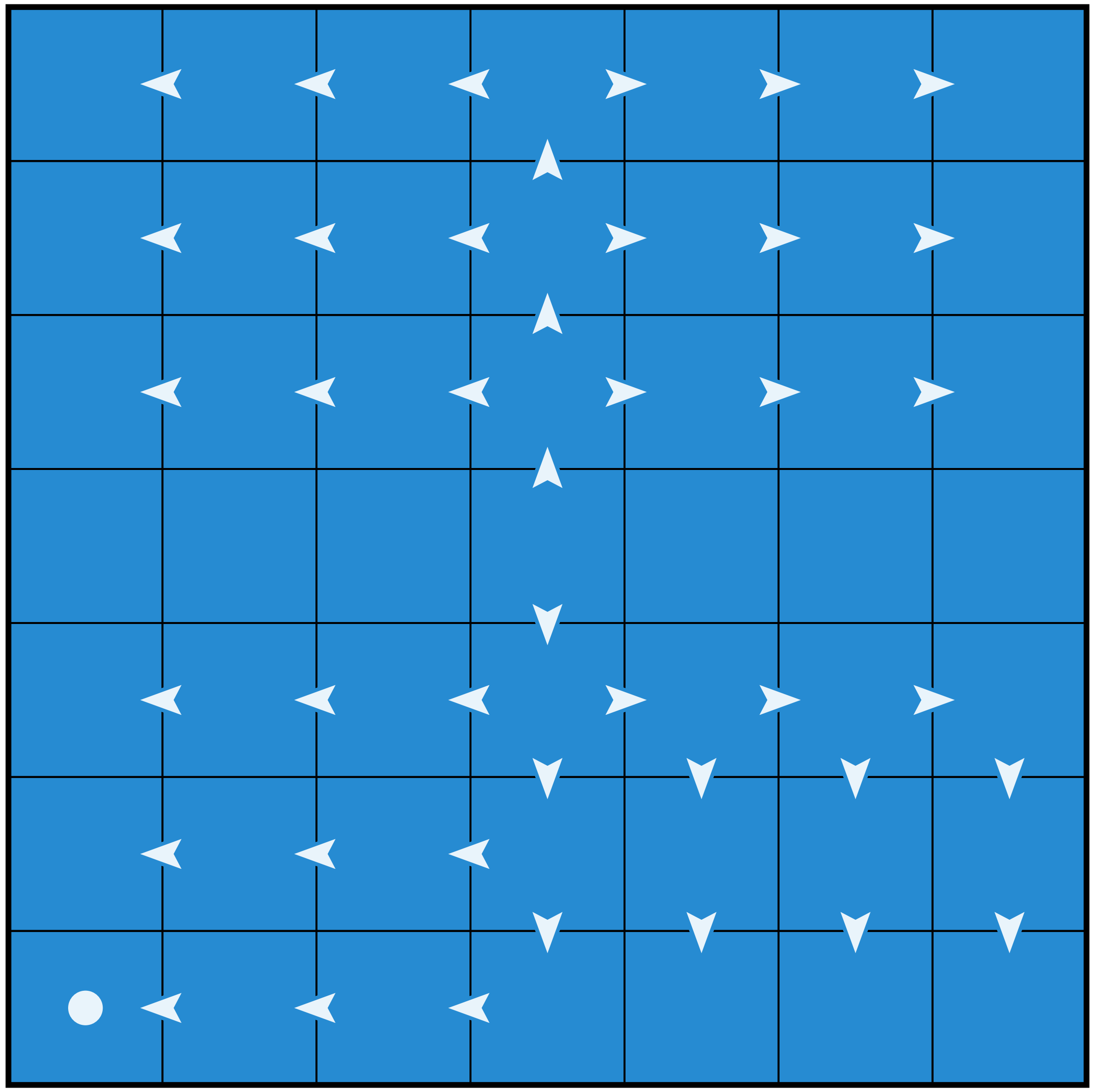
Traversering › BFS

Opprinnelig beskrevet av Konrad Zuse i 1945. Publisert først i 1972 (og i mellomtiden oppdaget av flere).

Traversering generelt:
Vi besøker noder, oppdager noder langs kanter og vedlikeholder en huskeliste.



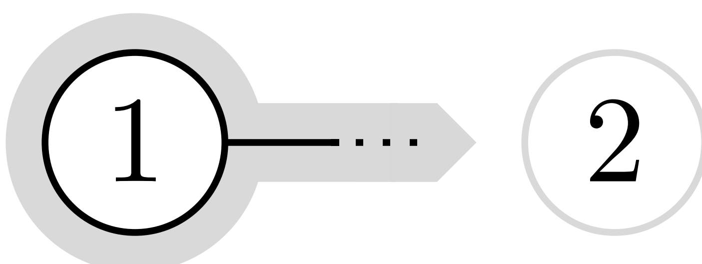
BFS: Naboer stiller seg i kø



BFS: Naboer stiller seg i kø

Besøk node 1

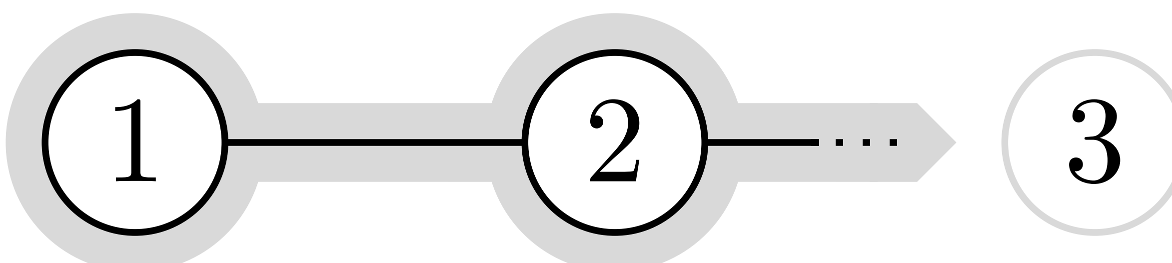
Har notert oss en startnode



~~Besøk node 1~~

Besøk node 2

Besøk: Stryk noden og notér naboer

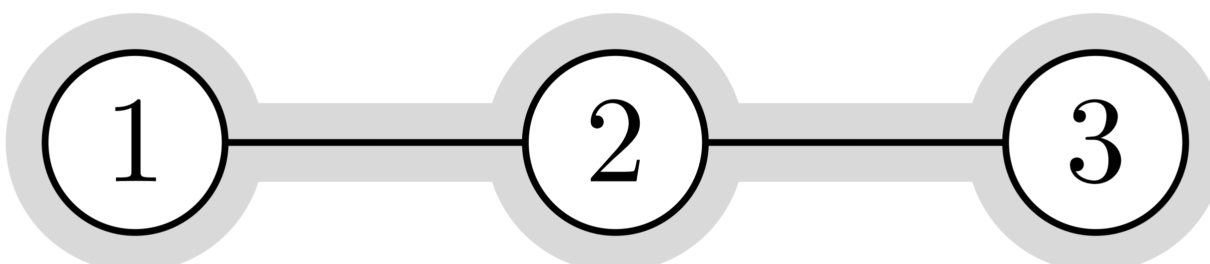


~~Besøk node 1~~

~~Besøk node 2~~

Besøk node 3

Besøk deretter neste på lista . . .



~~Besøk node 1~~

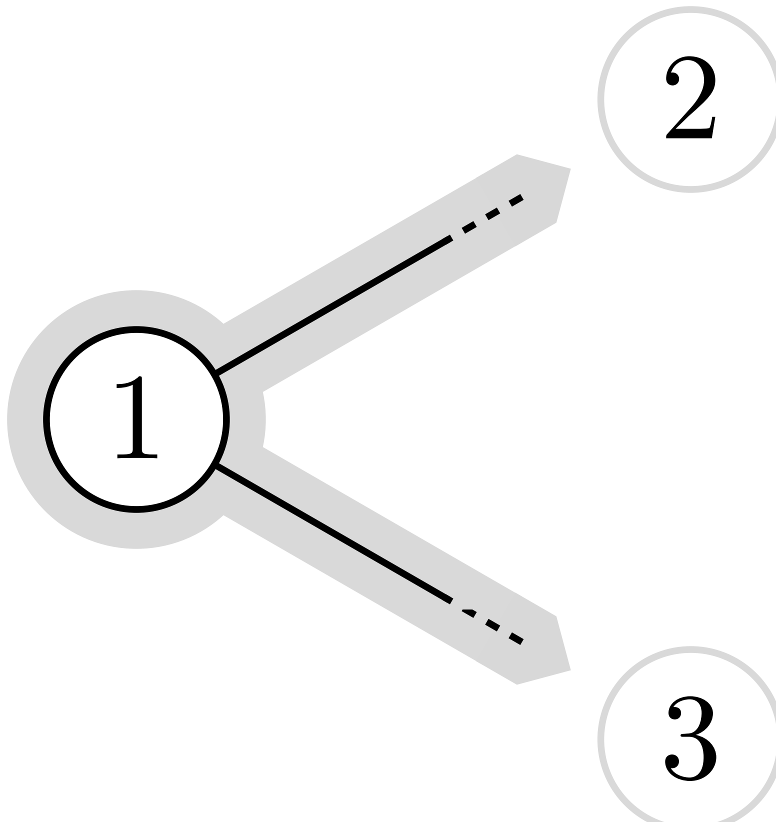
~~Besøk node 2~~

~~Besøk node 3~~

. . . helt til lista er tom; har besøkt alle vi kan

Besøk node 1

Har notert oss en startnode

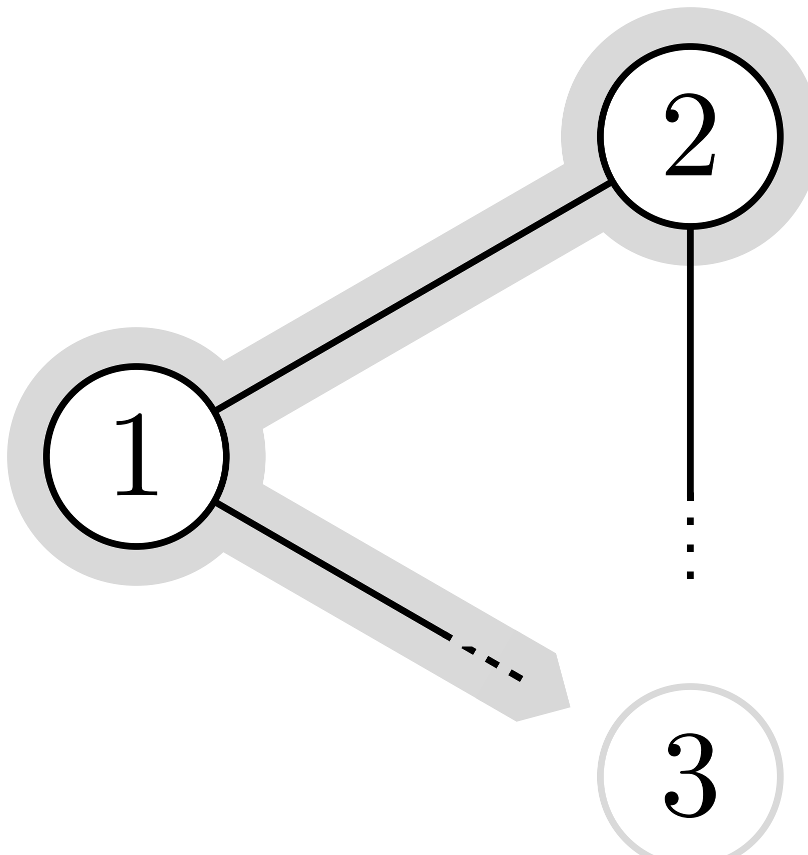


~~Besøk node 1~~

Besøk node 2

Besøk node 3

Besøk: Stryk noden og notér naboer

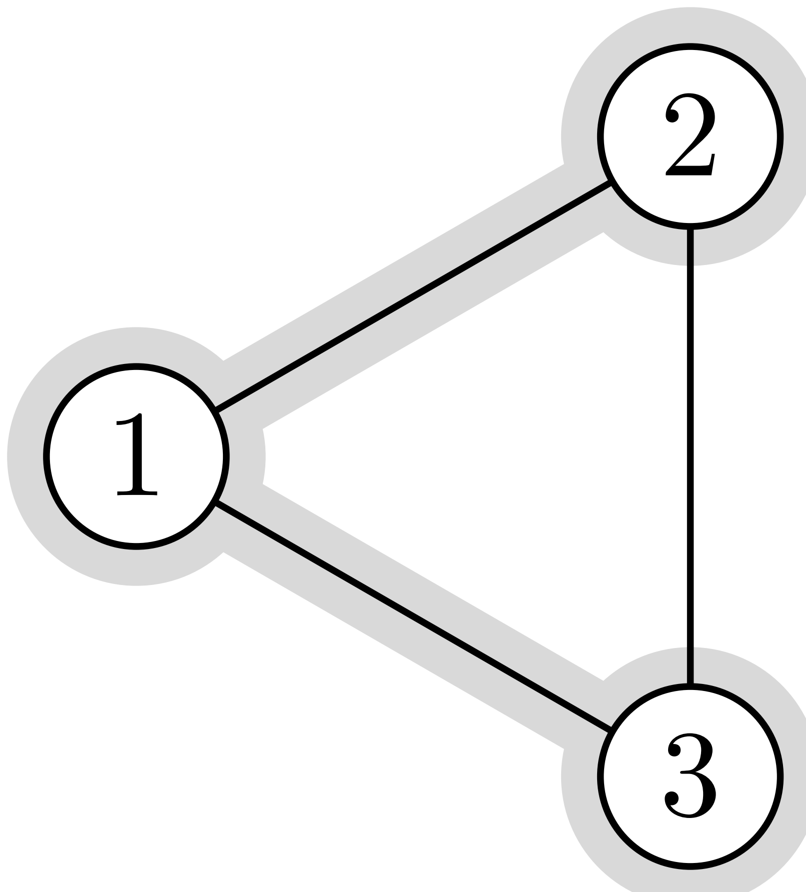


~~Besøk node 1~~

~~Besøk node 2~~

Besøk node 3

Besøk deretter neste på lista . . .

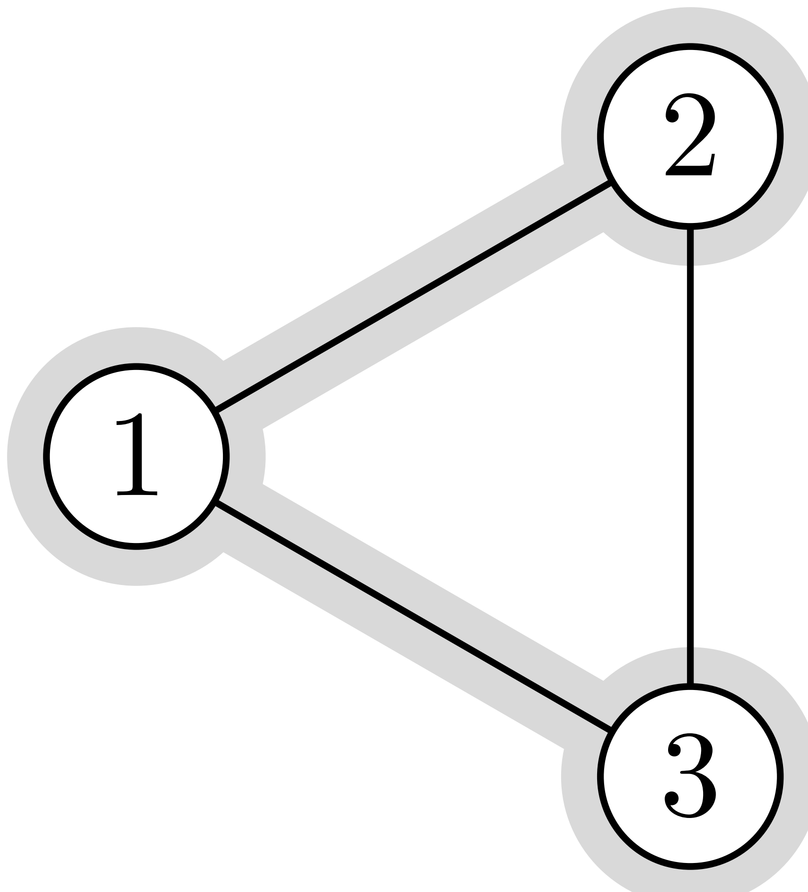


~~Besøk node 1~~

~~Besøk node 2~~

~~Besøk node 3~~

…helt til lista er tom; har besøkt alle vi kan

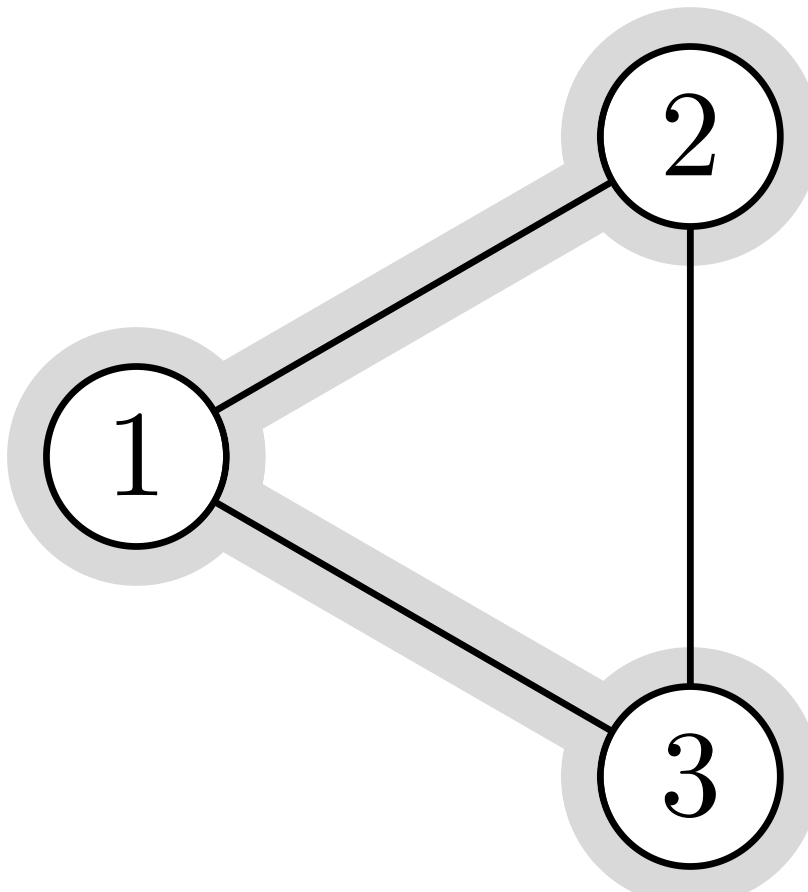


~~Besøk node 1~~

~~Besøk node 2~~

~~Besøk node 3~~

Uthevet: Forgjenger ($v.\pi$) for hver node v

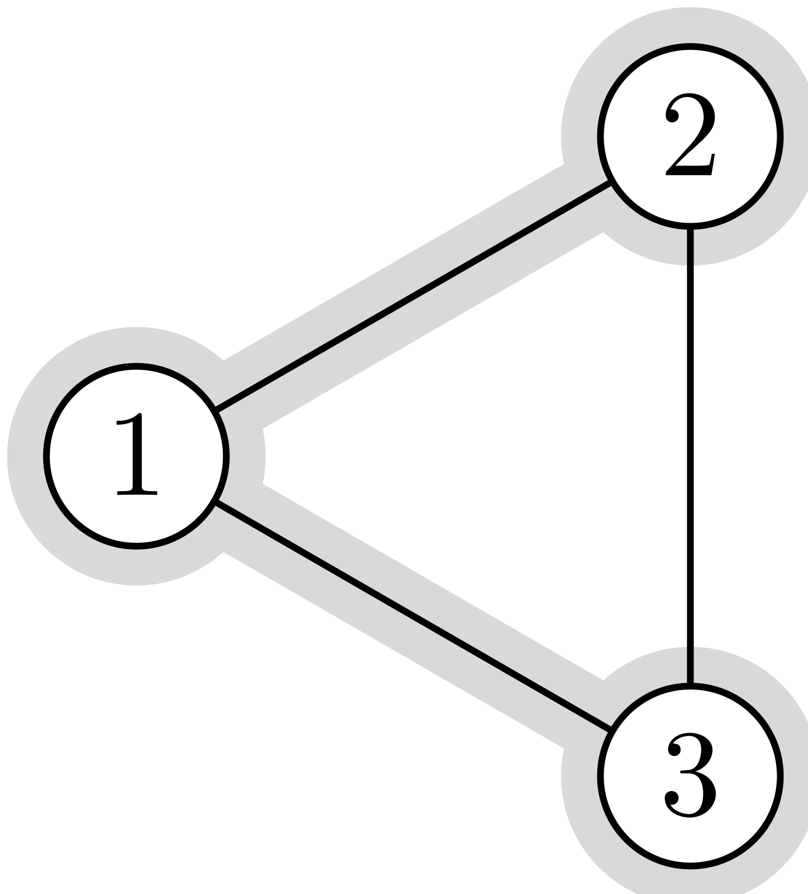


~~Besøk node 1~~

~~Besøk node 2~~

~~Besøk node 3~~

$v.\pi$: Hvilken node besøkte vi da vi oppdaget v?

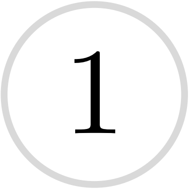


~~Besøk node 1~~

~~Besøk node 2~~

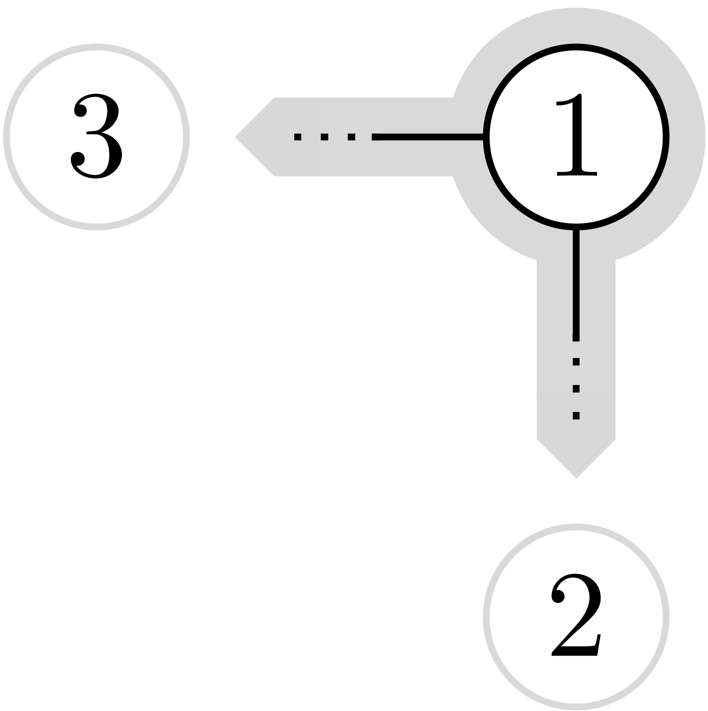
~~Besøk node 3~~

Forgjengerne utgjør *traverseringstreet*



Besøk node 1

Har notert oss en startnode

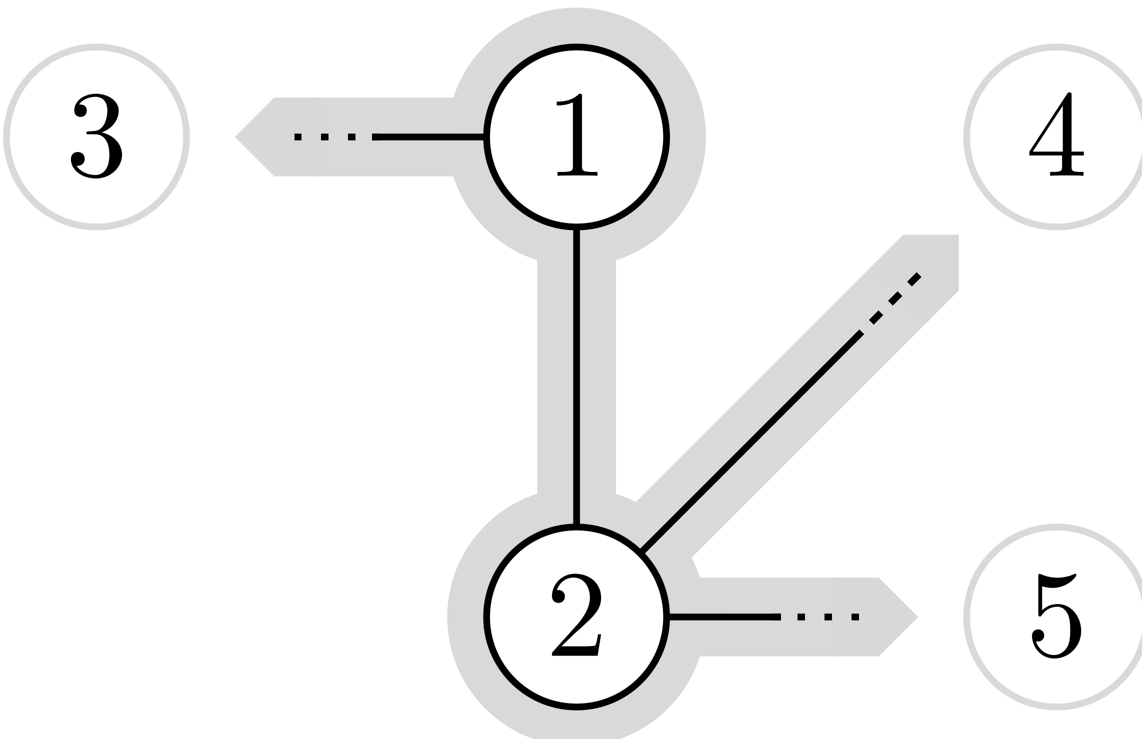


~~Besøk node 1~~

Besøk node 2

Besøk node 3

Besøk: Stryk noden og notér naboer



~~Besøk node 1~~

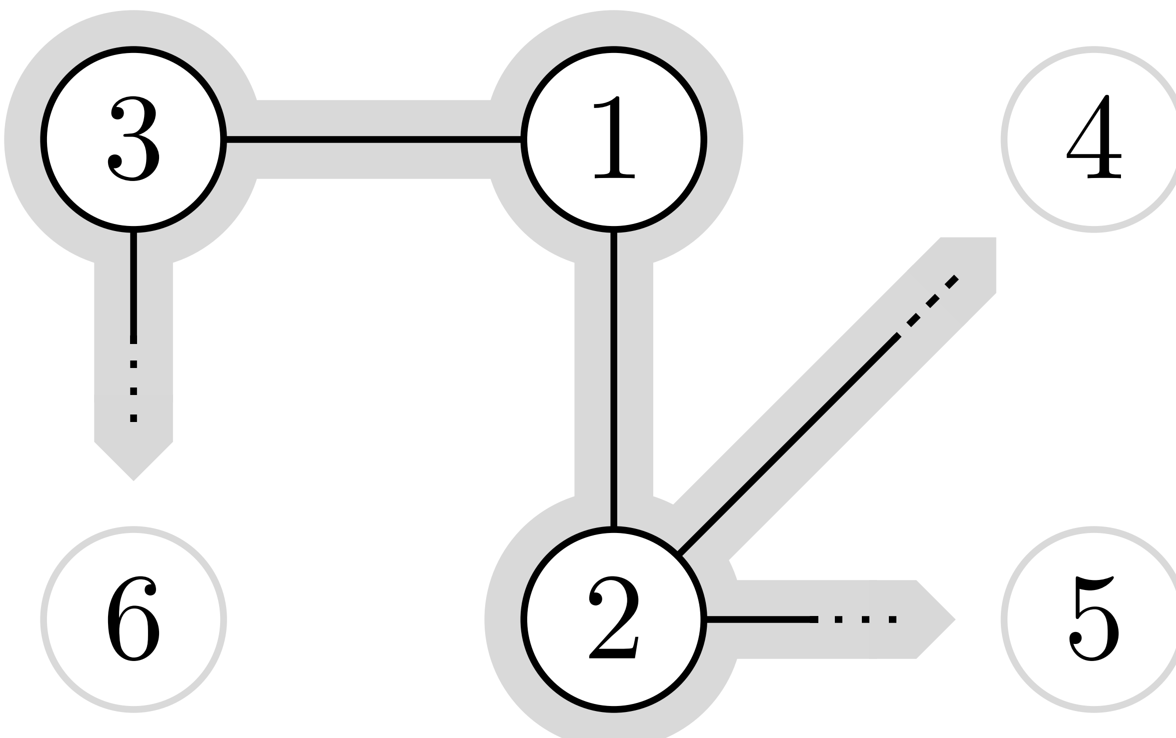
~~Besøk node 2~~

Besøk node 3

Besøk node 4

Besøk node 5

Besøk deretter neste på lista . . .



~~Besøk node 1~~

~~Besøk node 2~~

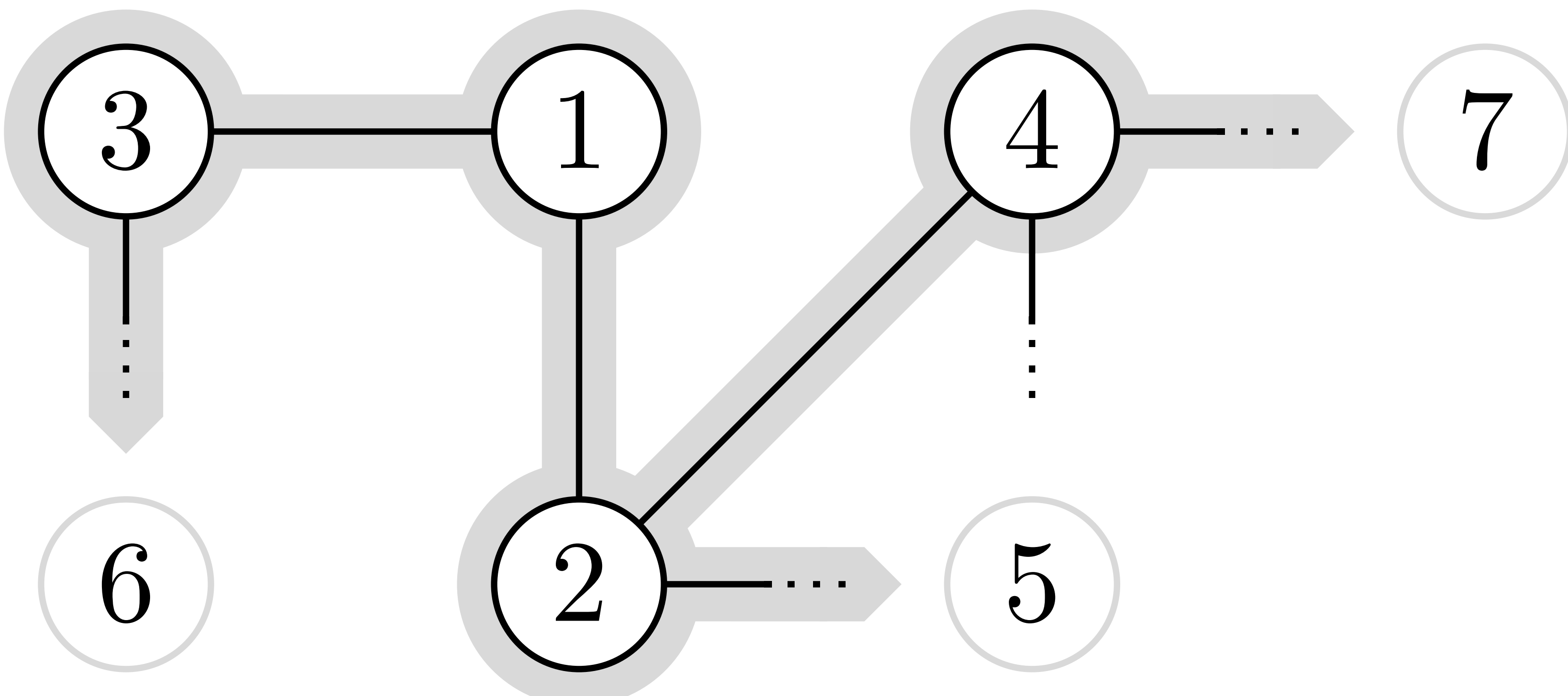
~~Besøk node 3~~

Besøk node 4

Besøk node 5

Besøk node 6

. . . og den neste . . .



~~Besøk node 1~~

~~Besøk node 2~~

~~Besøk node 3~~

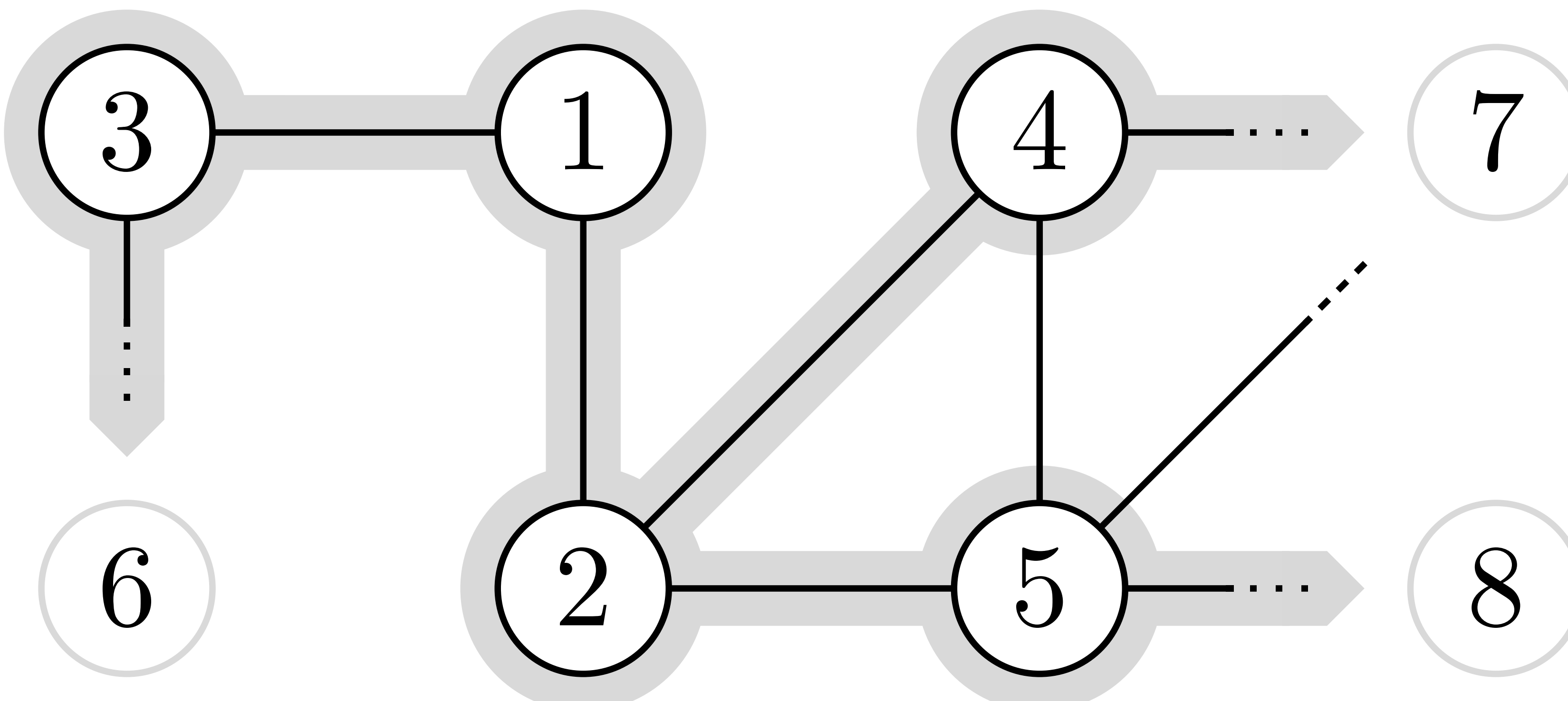
~~Besøk node 4~~

Besøk node 5

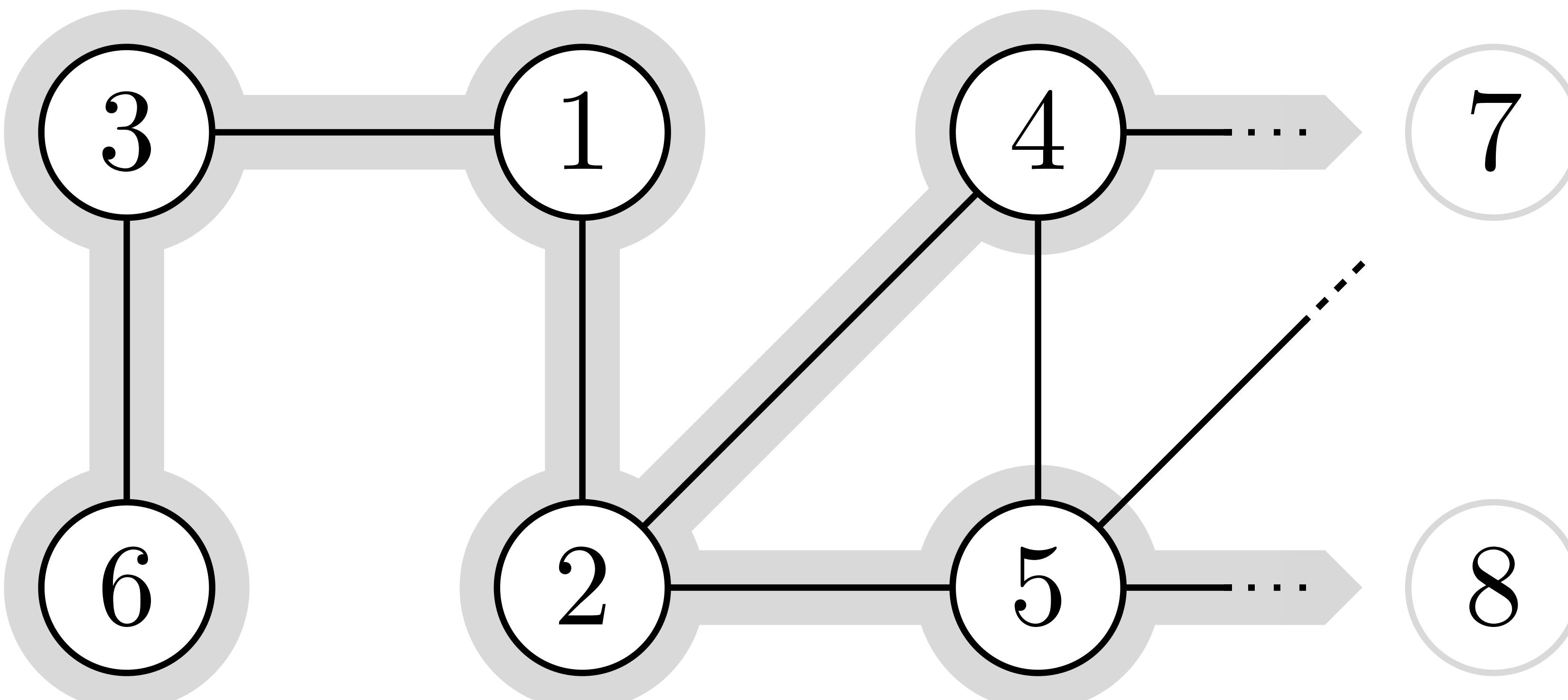
Besøk node 6

Besøk node 7

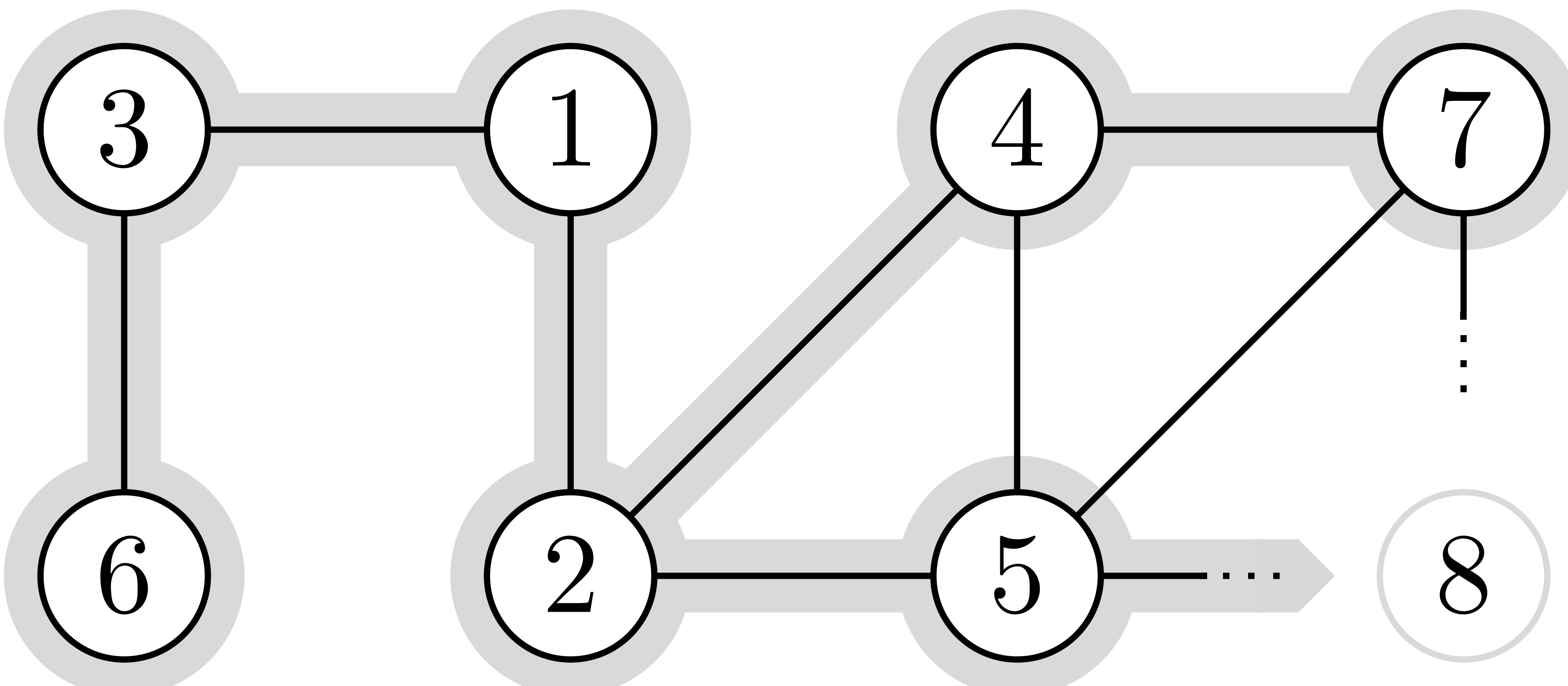
. . . og den neste . . .



~~Besøk node 1~~
~~Besøk node 2~~
~~Besøk node 3~~
~~Besøk node 4~~
~~Besøk node 5~~
Besøk node 6
Besøk node 7
Besøk node 8



~~Besøk node 1~~
~~Besøk node 2~~
~~Besøk node 3~~
~~Besøk node 4~~
~~Besøk node 5~~
~~Besøk node 6~~
Besøk node 7
Besøk node 8



~~Besøk node 1~~

~~Besøk node 2~~

~~Besøk node 3~~

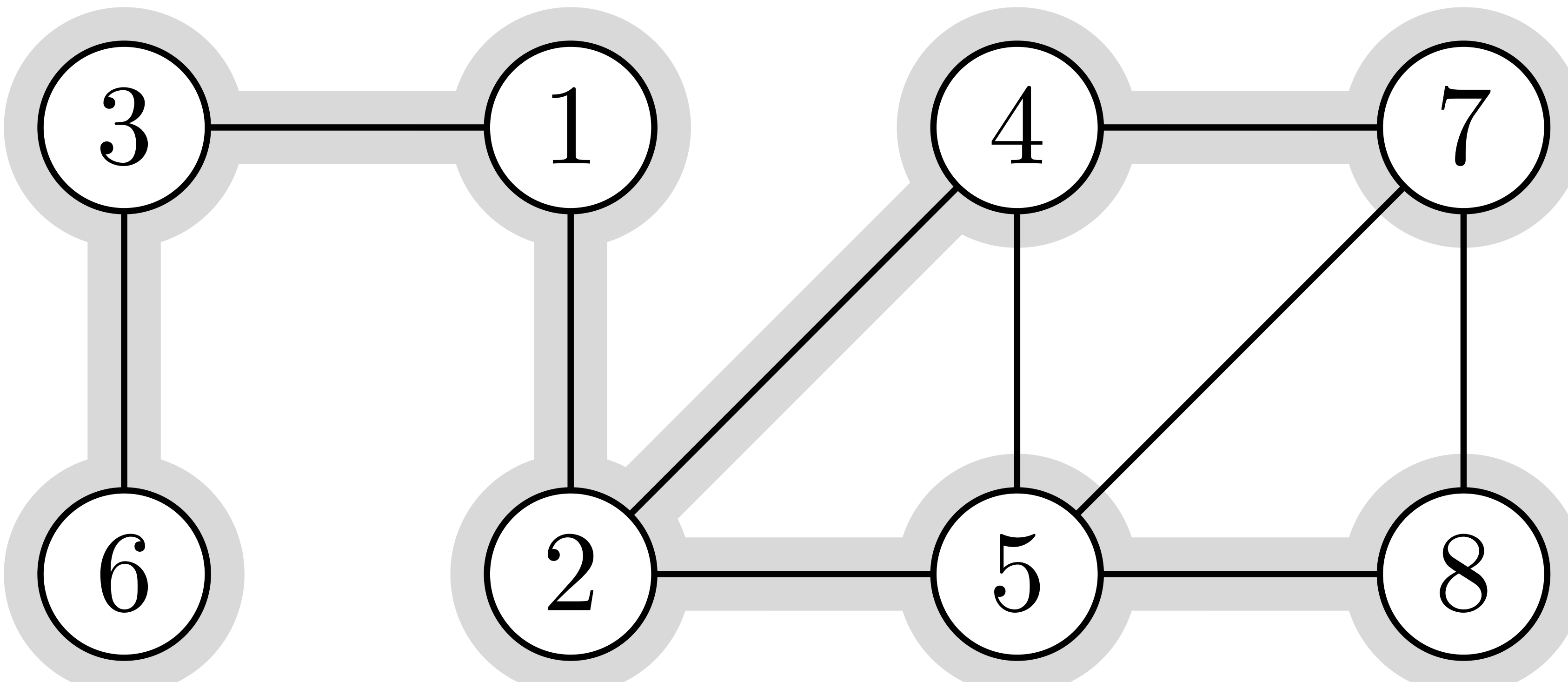
~~Besøk node 4~~

~~Besøk node 5~~

~~Besøk node 6~~

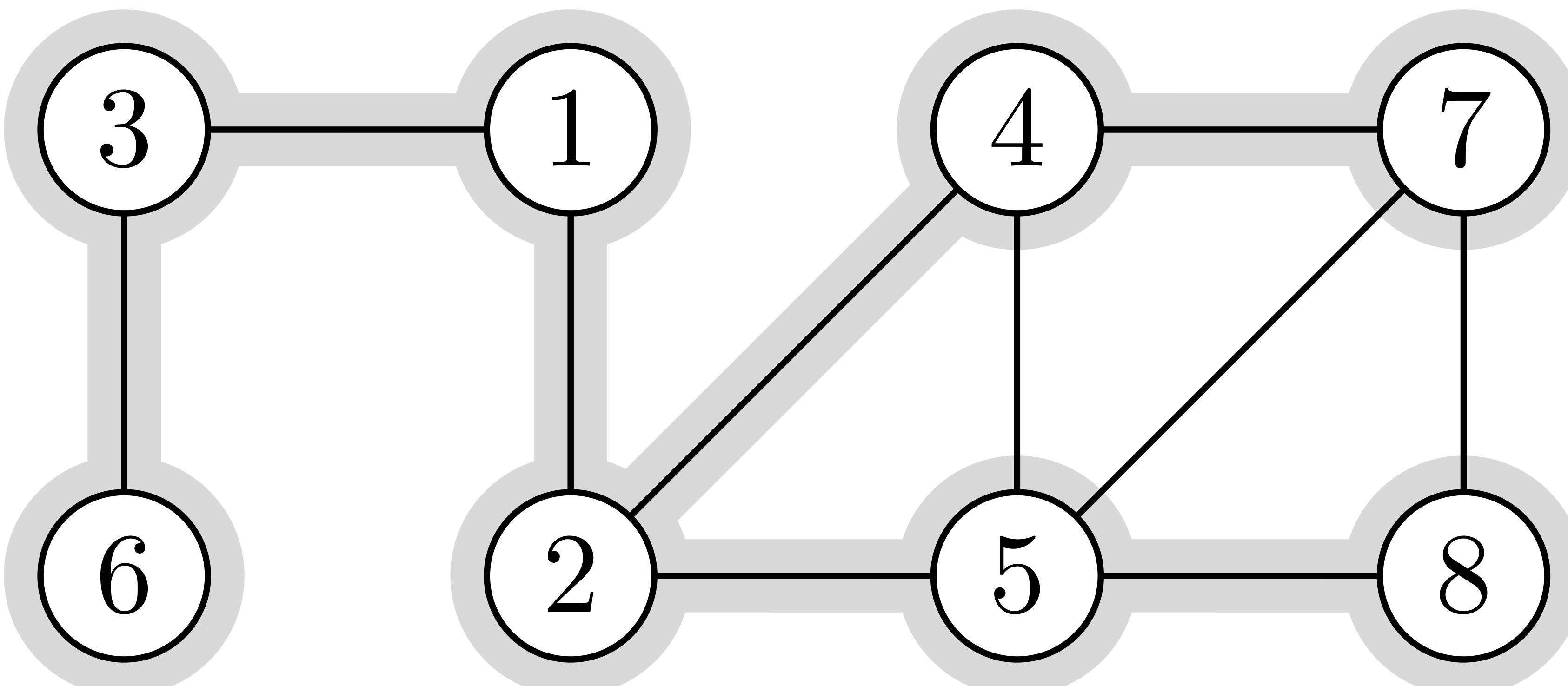
~~Besøk node 7~~

Besøk node 8



~~Besøk node 1~~
~~Besøk node 2~~
~~Besøk node 3~~
~~Besøk node 4~~
~~Besøk node 5~~
~~Besøk node 6~~
~~Besøk node 7~~
~~Besøk node 8~~

…helt til lista er tom; har besøkt alle vi kan



~~Besøk node 1~~

~~Besøk node 2~~

~~Besøk node 3~~

~~Besøk node 4~~

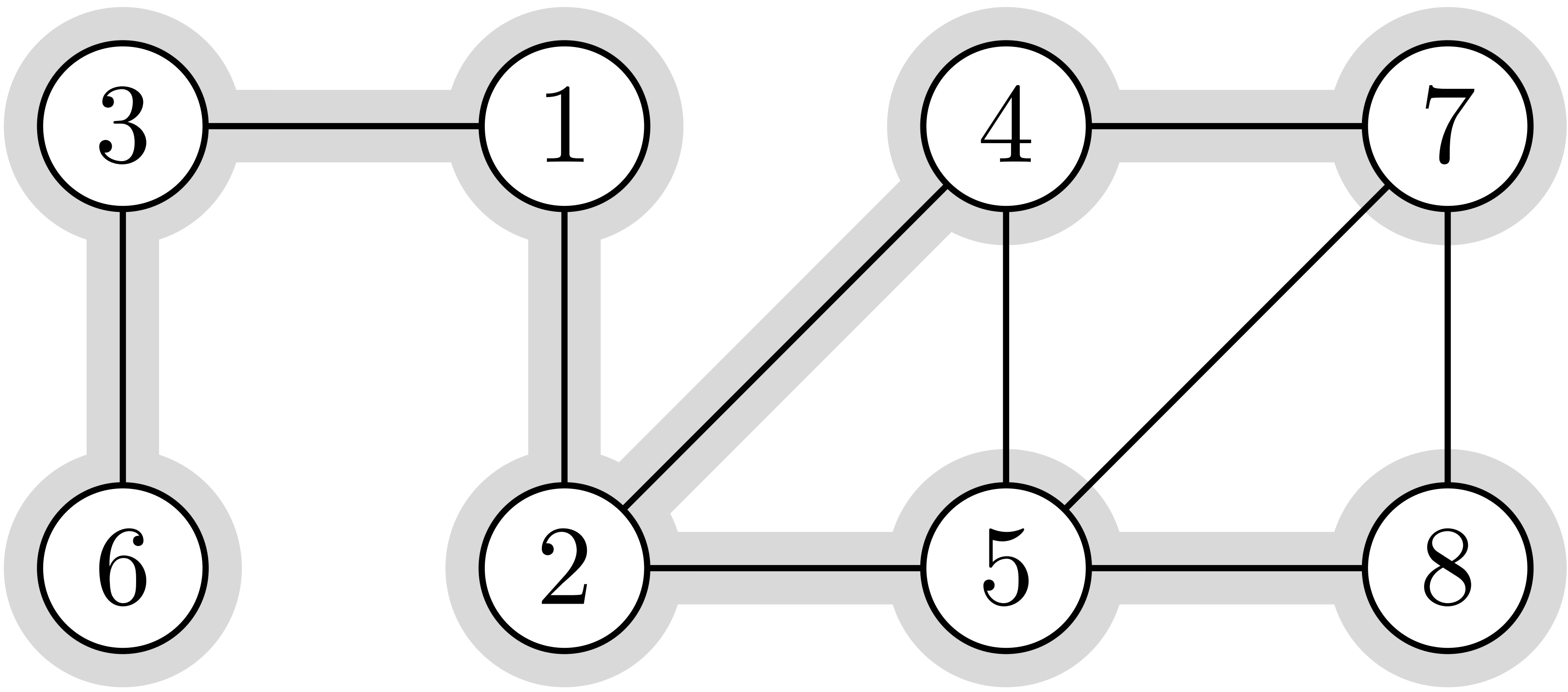
~~Besøk node 5~~

~~Besøk node 6~~

~~Besøk node 7~~

~~Besøk node 8~~

Traverseringstre uthevet: Hvor kom vi fra?



~~Besøk node 1~~
~~Besøk node 2~~
~~Besøk node 3~~
~~Besøk node 4~~
~~Besøk node 5~~
~~Besøk node 6~~
~~Besøk node 7~~
~~Besøk node 8~~

F.eks., $2 \to 5 \to 8$: $8.\pi = 5$, $5.\pi = 2$, $2.\pi = \textsc{nil}$

Så lenge vi bruker en FIFO-kø (dvs., BFS) så finner vi korteste vei; ellers risikerer vi å finne noder via omveier!

Boka har et relativt rett frem (om noe omstendelig) bevis for at BFS finner korteste vei.

BFS(G, s)

G graf

s startnode

Traversér noder oppdaget fra s, så noder oppdaget fra disse, etc.

BFS(G, s)
1 **for** each vertex $u \in$ G.V $- \{s\}$

G graf
V noder
s startnode

Først initialisering . . .

BFS(G, s)
for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$

G	graf
V	noder
s	startnode

Hvit = uoppdaget

BFS(G, s)

for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$

G	graf
V	noder
s	startnode
d	avstand

Beste gjetning på avstand fra s

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
```

G	graf
V	noder
s	startnode
d	avstand
π	forgjenger

Hvilken node har vi oppdaget u fra?

BFS(G, s)

for each vertex $u \in \mathrm{G.V} - \{s\}$
$\quad u.color = \textsc{white}$
$\quad u.d = \infty$
$\quad u.\pi = \textsc{nil}$
$s.color = \textsc{gray}$

G	graf
V	noder
s	startnode
d	avstand
π	forgjenger

Grå = oppdaget men ikke besøkt

BFS(G, s)

for each vertex $u \in G.V - \{s\}$

    $u.color = \text{WHITE}$

    $u.d = \infty$

    $u.\pi = \text{NIL}$

$s.color = \text{GRAY}$

$s.d = 0$

G	graf
V	noder
s	startnode
d	avstand
π	forgjenger

Avstand fra s til s

BFS(G, s)

for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$

G graf
V noder
s startnode
d avstand
π forgjenger

Startnoden har aldri noen forgjenger

BFS(G, s)

for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$

G	graf
V	noder
s	startnode
d	avstand
π	forgjenger
Q	kø

«Huskelisten»: En FIFO-kø. Oppdaget tidlig $\Longrightarrow$ besøkt tidlig

BFS(G, s)

for each vertex $u \in G.V - \{s\}$
  $u.color = \text{WHITE}$
  $u.d = \infty$
  $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
ENQUEUE(Q, s)

G	graf
V	noder
s	startnode
d	avstand
π	forgjenger
Q	kø

Før vi starter: «Skriv opp» startnoden på huskelisten

BFS(G, s)

for each vertex $u \in G.V - \{s\}$

  $u.color = \textsc{white}$

  $u.d = \infty$

  $u.\pi = \textsc{nil}$

$s.color = \textsc{gray}$

$s.d = 0$

$s.\pi = \textsc{nil}$

$Q = \emptyset$

$\textsc{Enqueue}(Q, s)$

$\ldots$

G	graf
V	noder
s	startnode
d	avstand
π	forgjenger
Q	kø

mlh@ntnu.no

BFS(G, s)

…

G graf
s startnode

BFS(G, s)
…
while $Q \neq \emptyset$

G graf
s startnode
Q kø

Så lenge vi har oppdagede, ubesøkte noder …

BFS(G, s)

...

while Q $\neq \emptyset$

$\quad u = \textsc{Dequeue}(\mathrm{Q})$

G graf
s startnode
Q kø
u besøkes

... velg den av dem vi oppdaget først

```
BFS(G, s)
    ...
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
```

G	graf
s	startnode
Q	kø
u	besøkes
v	nabonode

For hver av naboene til den besøkte noden . . .

BFS(G, s)

...

while $\mathrm{Q} \neq \emptyset$

    $u = \textsc{Dequeue}(\mathrm{Q})$

    for each $v \in \mathrm{G}.Adj[u]$

        if $v.color == \textsc{white}$

G graf
s startnode
Q kø
u besøkes
v nabonode

Er noden uoppdaget?

```
BFS(G, s)
    ...
    while Q ≠ ∅
        u = Dequeue(Q)
        for each v ∈ G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
```

G graf
s startnode
Q kø
u besøkes
v nabonode

Ikke nå lenger!

BFS(G, s)

```
...
while Q ≠ ∅
    u = Dequeue(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
```

G	graf
s	startnode
Q	kø
u	besøkes
v	nabonode
d	avstand

$$s \rightsquigarrow v = s \rightsquigarrow u \rightarrow v$$

BFS(G, s)

```
...
while Q ≠ ∅
    u = Dequeue(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
```

G graf
s startnode
Q kø
u besøkes
v nabonode
d avstand
π forgjenger

$$s \rightsquigarrow v = s \rightsquigarrow u \rightarrow v$$

```
BFS(G, s)
...
while Q ≠ ∅
    u = Dequeue(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            Enqueue(Q, v)
```

G	graf
s	startnode
Q	kø
u	besøkes
v	nabonode
d	avstand
π	forgjenger

Vi må huske å besøke den nyoppdagede noden etter hvert

```
BFS(G, s)
  ...
  while Q ≠ ∅
      u = DEQUEUE(Q)
      for each v ∈ G.Adj[u]
          if v.color == WHITE
              v.color = GRAY
              v.d = u.d + 1
              v.π = u
              ENQUEUE(Q, v)
      u.color = BLACK
```

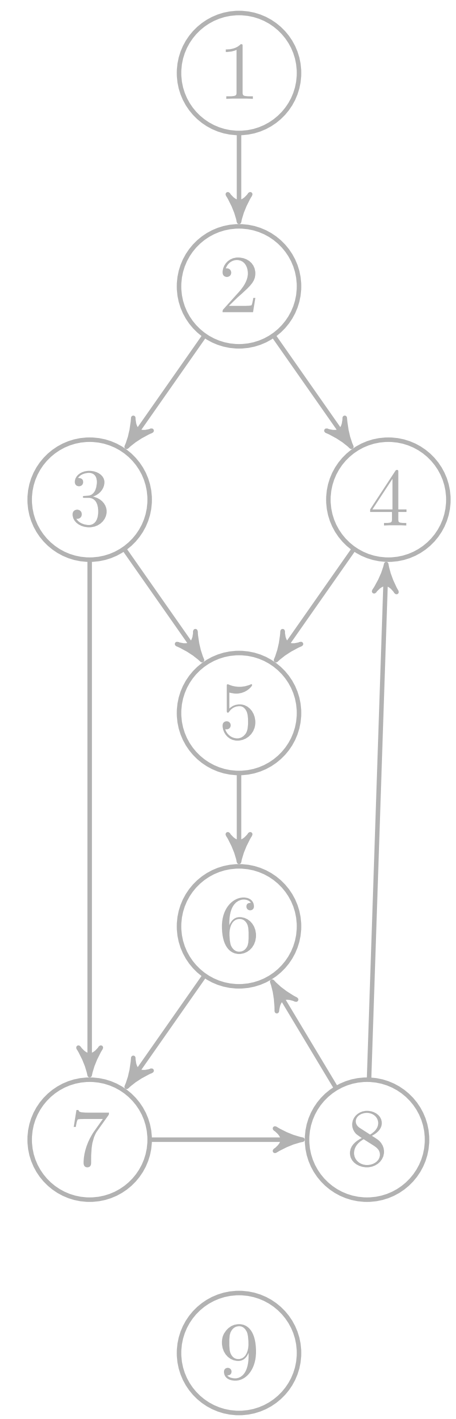
G graf
s startnode
Q kø
u besøkes
v nabonode
d avstand
π forgjenger

Svart = besøkt (og ferdigbehandlet)

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
...
```

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

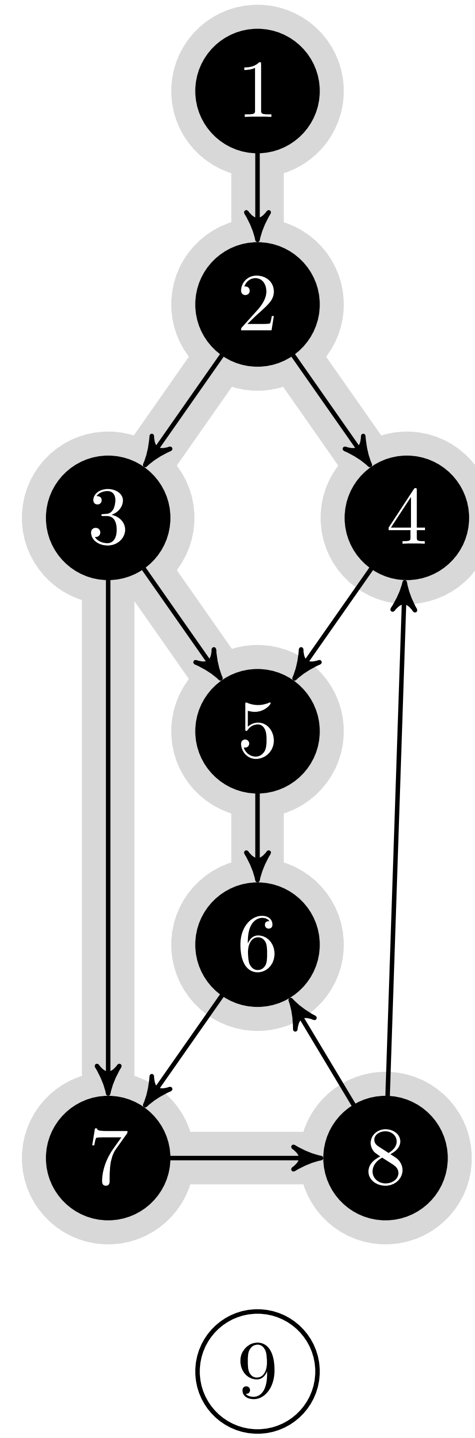
Q	d	π	
			1
			2
			3
			4
			5
			6
			7
			8
			9



```
BFS(G, s)
    ...
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.color = BLACK
```

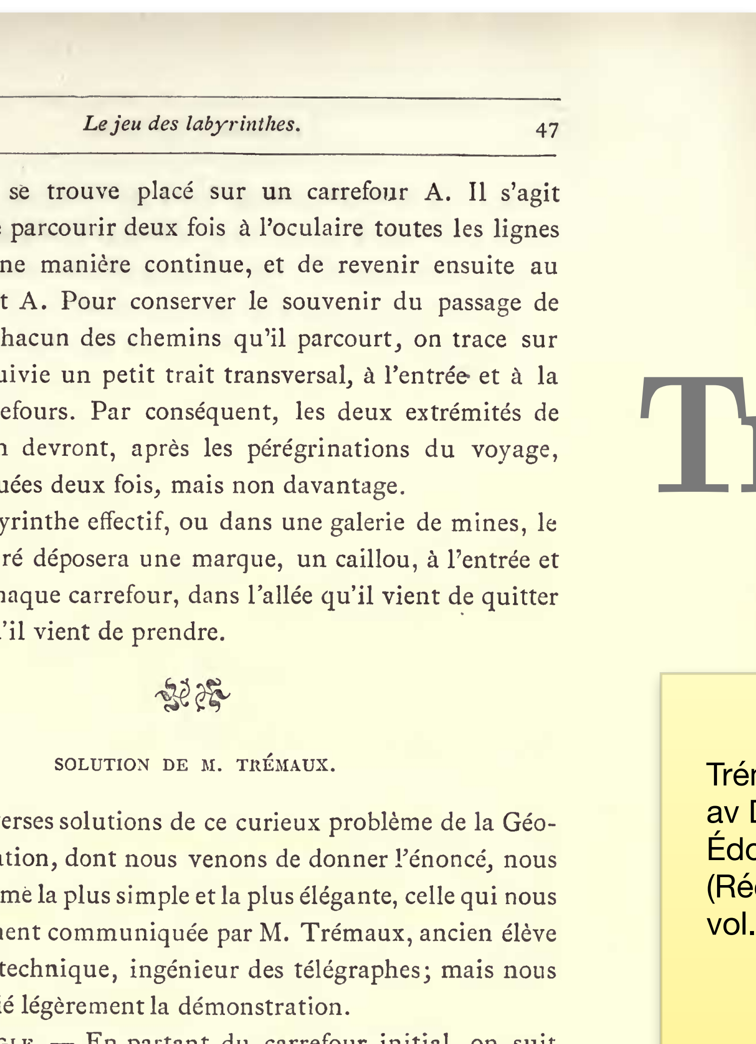
$u, v \;=\; -, -$

	Q	d	π	
	1	0	–	1
	2	1	1	2
	3	2	2	3
	4	2	2	4
	7	3	3	5
	5	4	5	6
	8	3	3	7
	6	4	7	8
h, t		∞	–	9



3:4

Traversering › DFS

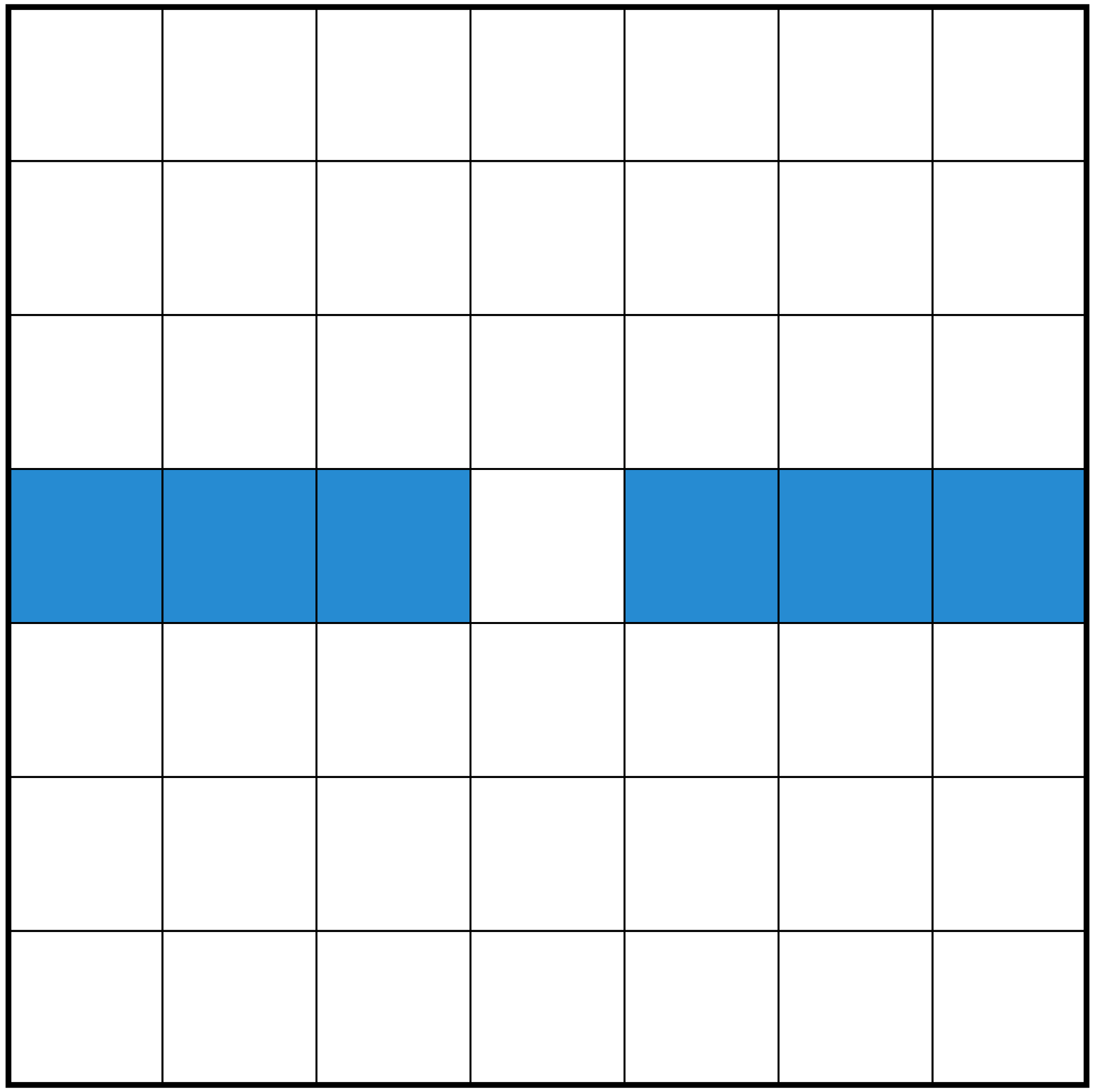

Le jeu des labyrinthes. 47

se trouve placé sur un carrefour A. Il s'agit
parcourir deux fois à l'oculaire toutes les lignes
ne manière continue, et de revenir ensuite au
t A. Pour conserver le souvenir du passage de
hacun des chemins qu'il parcourt, on trace sur
uivie un petit trait transversal, à l'entrée et à la
efours. Par conséquent, les deux extrémités de
n devront, après les pérégrinations du voyage,
uées deux fois, mais non davantage.
yrinthe effectif, ou dans une galerie de mines, le
ré déposera une marque, un caillou, à l'entrée et
haque carrefour, dans l'allée qu'il vient de quitter
'il vient de prendre.

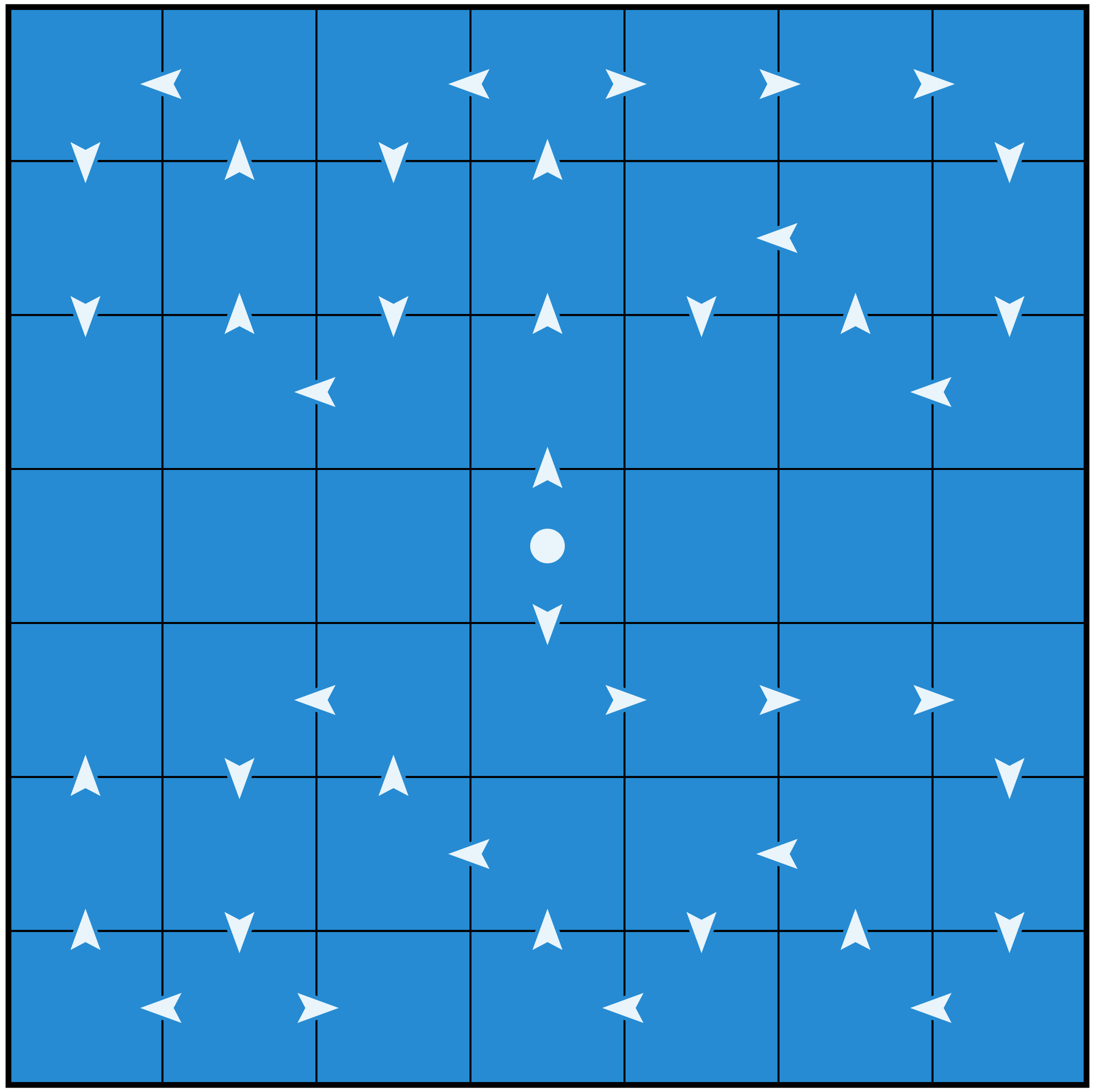
SOLUTION DE M. TRÉMAUX.

erses solutions de ce curieux problème de la Géo-
tion, dont nous venons de donner l'énoncé, nous
me la plus simple et la plus élégante, celle qui nous
ent communiquée par M. Trémaux, ancien élève
technique, ingénieur des télégraphes; mais nous
é légèrement la démonstration.

Trémaux's algoritme, en versjon av DFS, er her beskrevet av Édouard Lucas i 1882 (Récréations Mathématiques, vol. I).

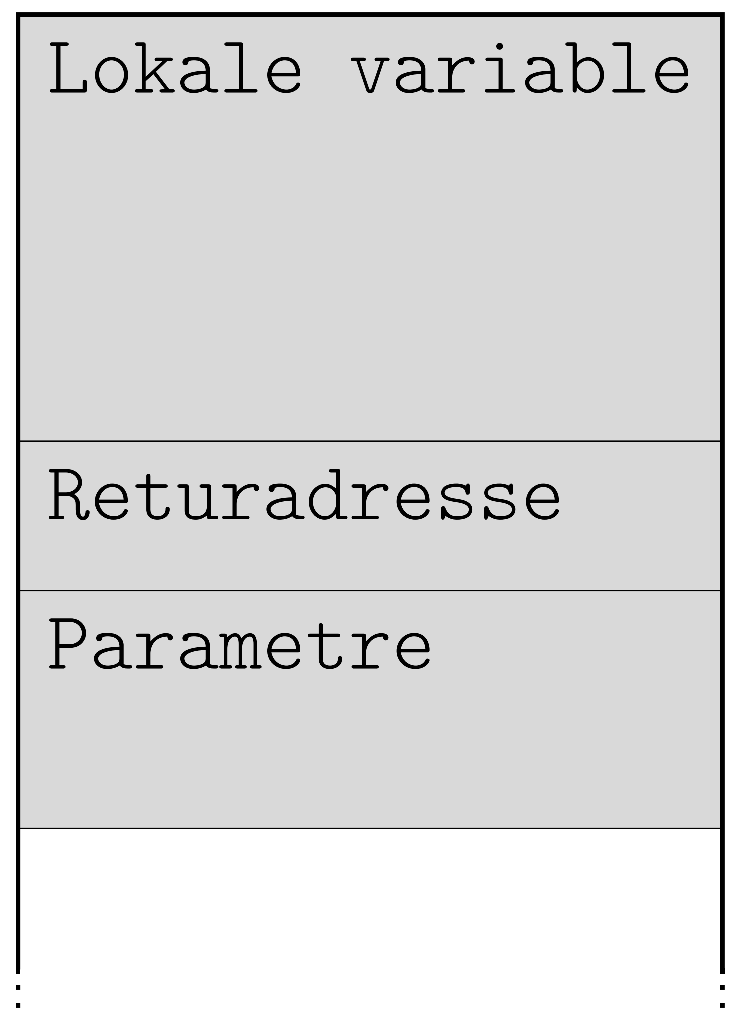


DFS, *flood-fill*: Fyll rekursivt nord, øst, sør, vest



DFS, *flood-fill*: Fyll rekursivt nord, øst, sør, vest

- **Som BFS, men med LIFO-kø**
- **Enklere å implementere rekursivt**
- **LIFO-køen blir da i praksis kallstakken**



Tilstanden lagres midlertidig under funksjonskall

DFS(G)

G graf

Besøk oppdagede noder umiddelbart

DFS(G)
1 **for** each vertex $u \in$ G.V

G graf

Først initialisering . . .

DFS(G)

for each vertex $u \in G.V$

  $u.color = \text{WHITE}$

G graf

Hvit = uoppdaget

DFS(G)

```
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
```

G graf
π forgjenger

Hvilken node har vi oppdaget u fra?

```
DFS(G)
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0        › global
```

G graf
π forgjenger

Vi holder styr på når vi oppdaget og ferdigbehandlet hver node

DFS(G)

1 **for** each vertex $u \in$ G.V
2 $\quad u.color = \text{WHITE}$
3 $\quad u.\pi = \text{NIL}$
4 $time = 0$ › *global*
5 **for** each vertex $u \in$ G.V

G graf
π forgjenger

I denne implementasjonen: Traversér fra alle noder

```
DFS(G)
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0          ▷ global
for each vertex u ∈ G.V
        if u.color == WHITE
```

G graf
π forgjenger

Ignorér noder vi alt er ferdige med

DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$ › *global*
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)

G graf
π forgjenger

Dette er den faktiske traverseringen!

DFS-VISIT(G, u)

G graf
u startnode

Dybde-først-traversering fra u

DFS-VISIT(G, u)

$time = time + 1$

G graf

u startnode

Når oppdaget vi u (*discover-time*)?

DFS-VISIT(G, u)

1 $time = time + 1$
2 $u.d = time$

G graf
u startnode

d starttid

Når oppdaget vi u (*discover-time*)?

DFS-VISIT(G, u)

$time = time + 1$

$u.d = time$

$u.color =$ GRAY

G graf

u startnode

d starttid

u er oppdaget

mlh@ntnu.no

DFS-Visit(G, u)

```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
```

G	graf
u	startnode
v	nabonode
d	starttid

For hver nabo . . .

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
```

G	graf
u	startnode
v	nabonode
d	starttid

Er den uoppdaget?

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color =$ GRAY
for each $v \in G.Adj[u]$
    if $v.color ==$ WHITE
        $v.\pi = u$

G graf
u startnode
v nabonode
d starttid

π forgjenger

Vi oppdager den nå, fra u!

mlh@ntnu.no

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color =$ GRAY
for each $v \in G.Adj[u]$
    if $v.color ==$ WHITE
        $v.\pi = u$
        DFS-VISIT(G, v)

G graf
u startnode
v nabonode
d starttid

π forgjenger

Besøk u umiddelbart; kallstakken blir huskeliste!

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color =$ GRAY
for each $v \in G.Adj[u]$
  if $v.color ==$ WHITE
    $v.\pi = u$
    DFS-VISIT(G, v)
$u.color =$ BLACK

G graf
u startnode
v nabonode
d starttid

π forgjenger

Vi er ferdige med u

```
DFS-Visit(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-Visit(G, v)
u.color = BLACK
time = time + 1
```

G graf
u startnode
v nabonode
d starttid

π forgjenger

Når ble vi ferdige med u (*finish-time*)?

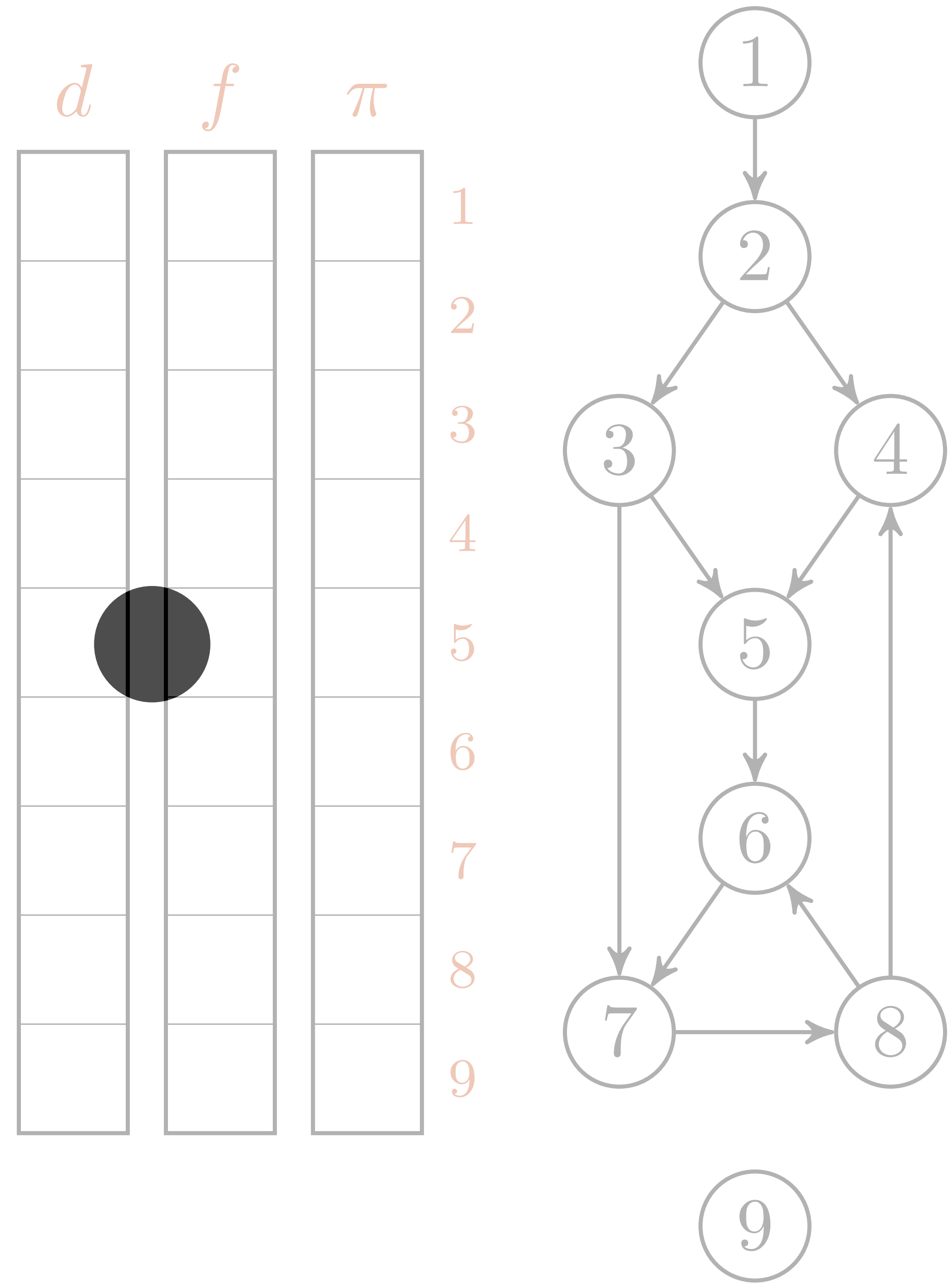
```
DFS-Visit(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-Visit(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

G graf
u startnode
v nabonode
d starttid
f sluttid
π forgjenger

Når ble vi ferdige med u (*finish-time*)?

DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)



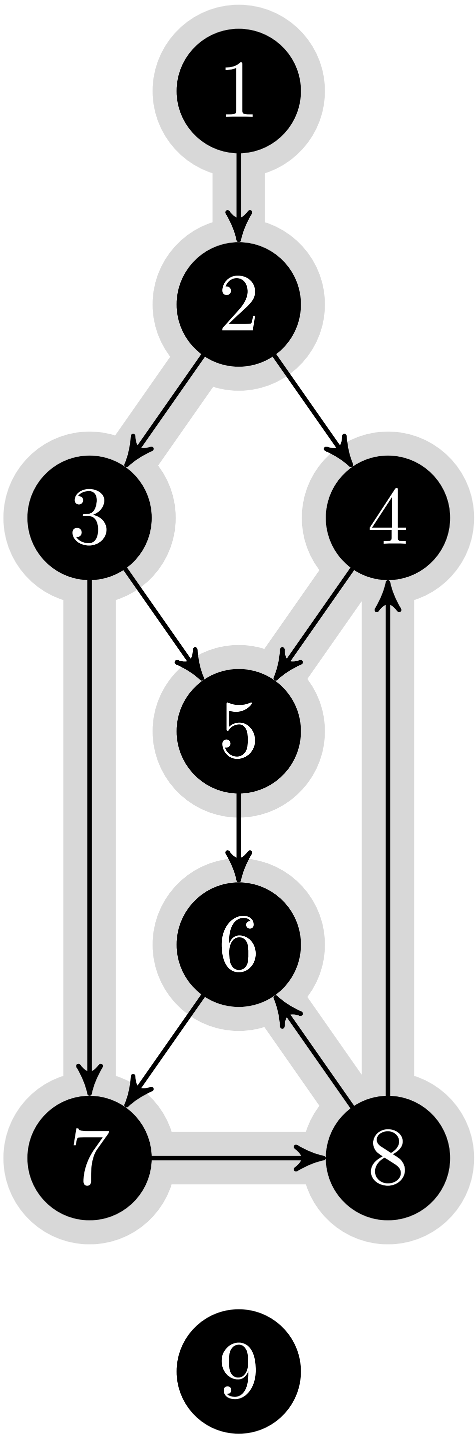
$u, v \;=\; \text{-}, \text{-}$

DFS(G)

```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-Visit(G, u)
```

$u, v = -, -$

	d	f	π
1	1	16	–
2	2	15	1
3	3	14	2
4	8	11	8
5	9	10	4
6	6	7	8
7	4	13	3
8	5	12	7
9	17	18	–



Kantklassifisering

- **Tre-kanter**
  Kanter i dybde-først-skogen
- **Bakoverkanter**
  Kanter til en forgjenger i DF-skogen
- **Foroverkanter**
  Kanter utenfor DF-skogen til en etterkommer i DF-skogen
- **Krysskanter**
  Alle andre kanter

Kantklassifisering

- **Møter en hvit node**

  Tre-kant

- **Møter en grå node**

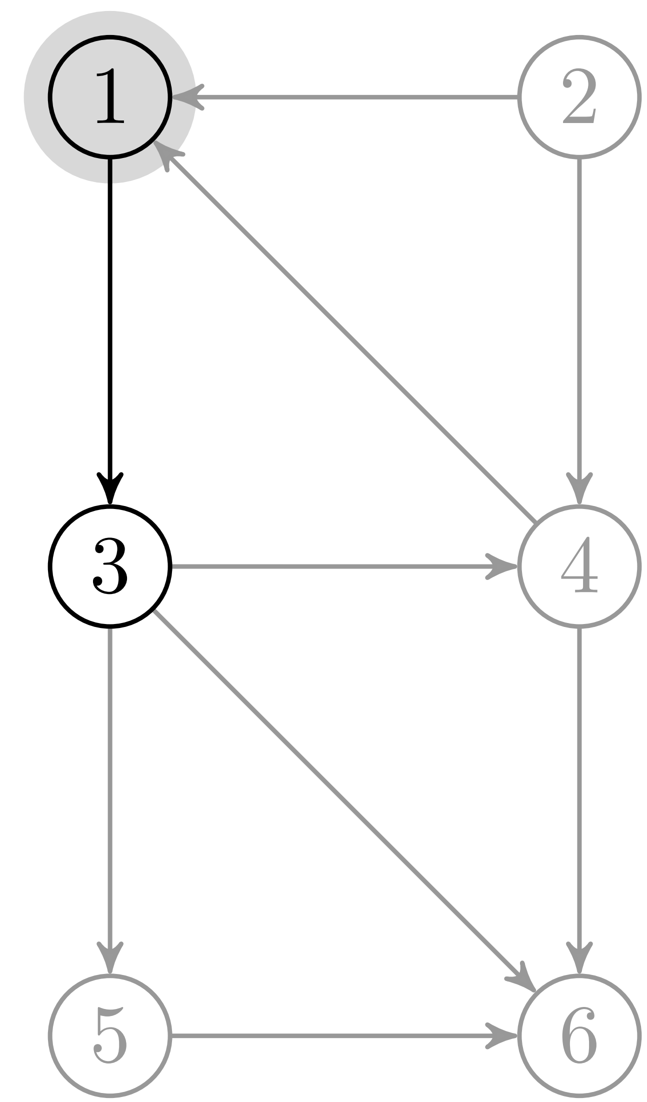
  Bakoverkant

- **Møter en svart node:**

  Forover- eller krysskant

```
DFS-VISIT(G, u)
  time = time + 1
  u.d = time
  u.color = GRAY
  for each v ∈ G.Adj[u]
      if v.color == WHITE
          v.π = u
          DFS-VISIT(G, v)
  u.color = BLACK
  time = time + 1
  u.f = time
```

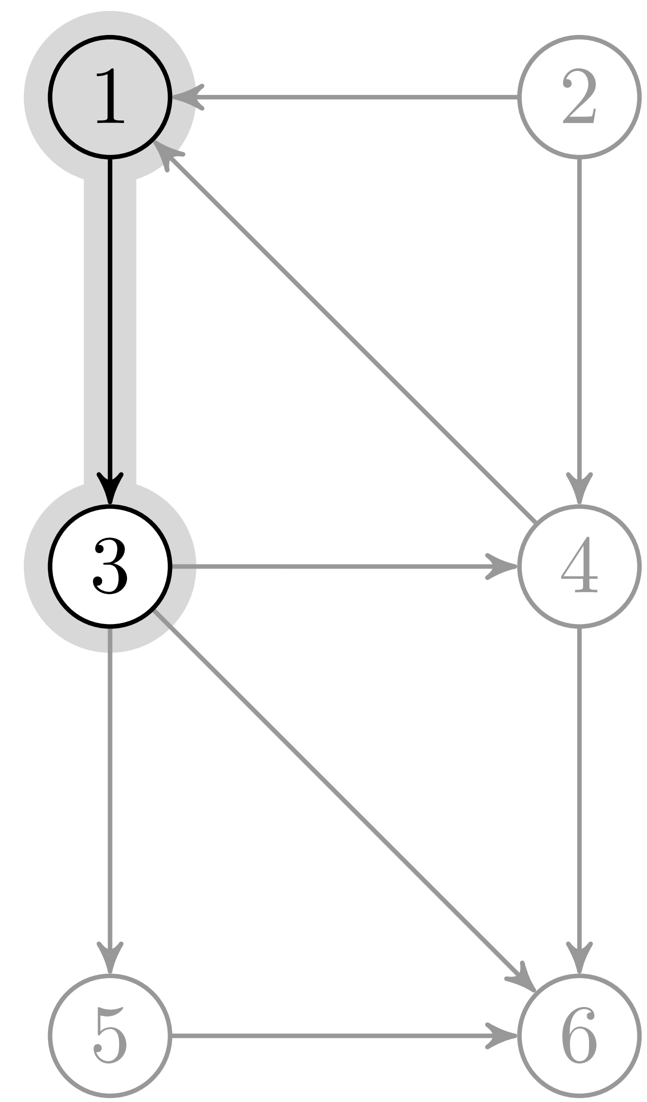
$u, v \;=\; 1, 3$



Vi oppdager v: Tre-kant

```
DFS-VISIT(G, u)
  time = time + 1
  u.d = time
  u.color = GRAY
  for each v ∈ G.Adj[u]
      if v.color == WHITE
          v.π = u
          DFS-VISIT(G, v)
  u.color = BLACK
  time = time + 1
  u.f = time
```

$u, v \;=\; 1, 3$

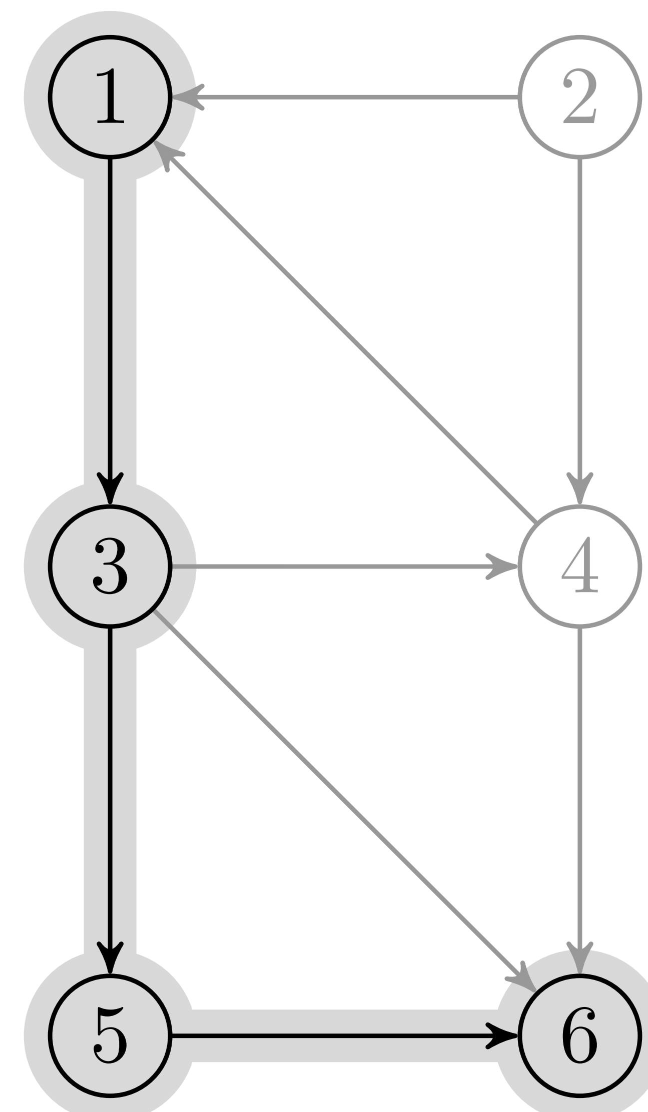


Vi oppdager v: Tre-kant

DFS-Visit(G, u)

$time = time + 1$
$u.d = time$
$u.color =$ GRAY
for each $v \in G.Adj[u]$
  if $v.color ==$ WHITE
    $v.\pi = u$
    DFS-Visit(G, v)
$u.color =$ BLACK
$time = time + 1$
$u.f = time$

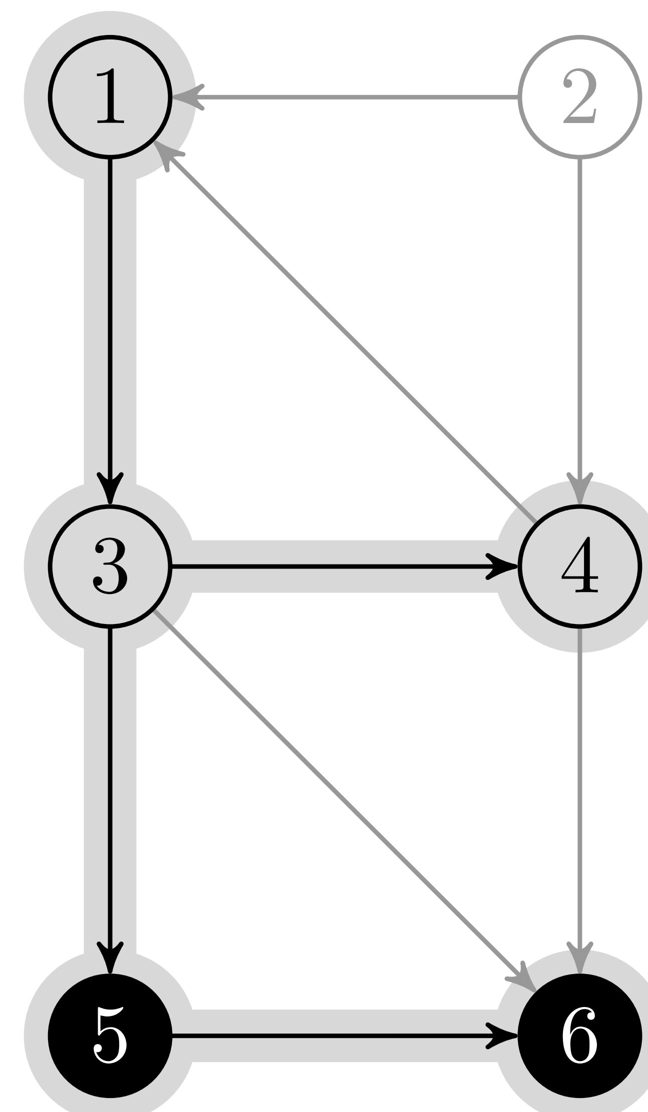
$u, v \;=\;$ 1, 3 › 3, 5 › 5, 6 › 6, -



Vi oppdager v: Tre-kant

```
DFS-Visit(G, u)
  time = time + 1
  u.d = time
  u.color = GRAY
  for each v ∈ G.Adj[u]
      if v.color == WHITE
          v.π = u
          DFS-Visit(G, v)
  u.color = BLACK
  time = time + 1
  u.f = time
```

$u, v \;=\; 1,3 \;›\; 3,4 \;›\; 4, \text{-}$

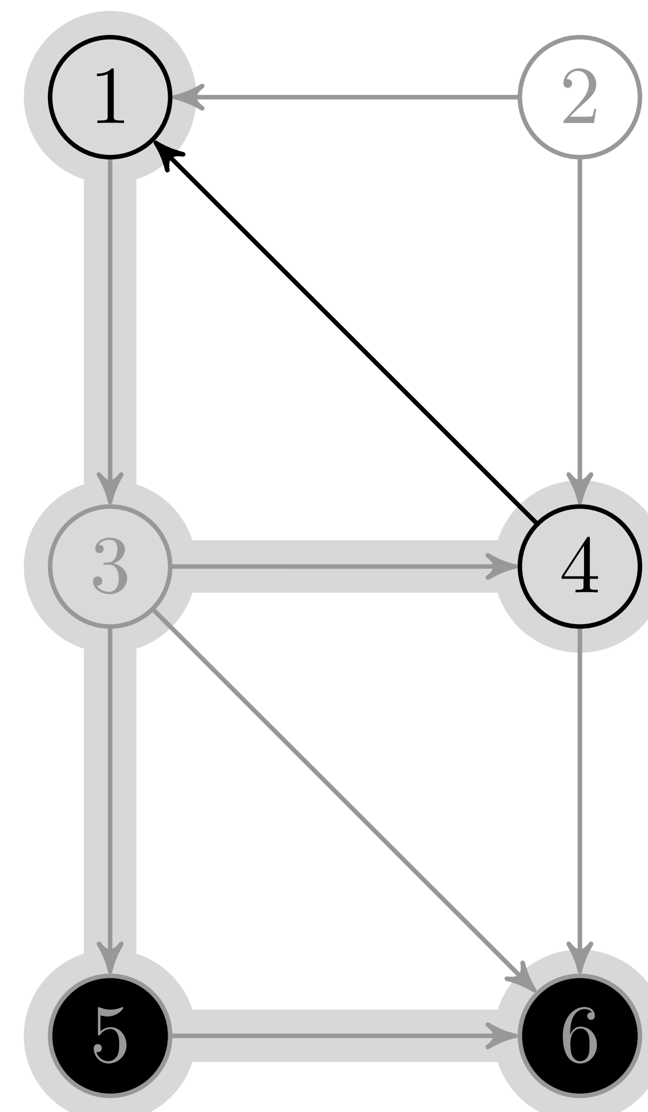


Tre-kanter utgjør DFS-treet

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color =$ GRAY
for each $v \in G.Adj[u]$
    if $v.color ==$ WHITE
        $v.\pi = u$
        DFS-VISIT(G, v)
$u.color =$ BLACK
$time = time + 1$
$u.f = time$

$u, v = 1, 3 \,›\, 3, 4 \,›\, 4, 1$

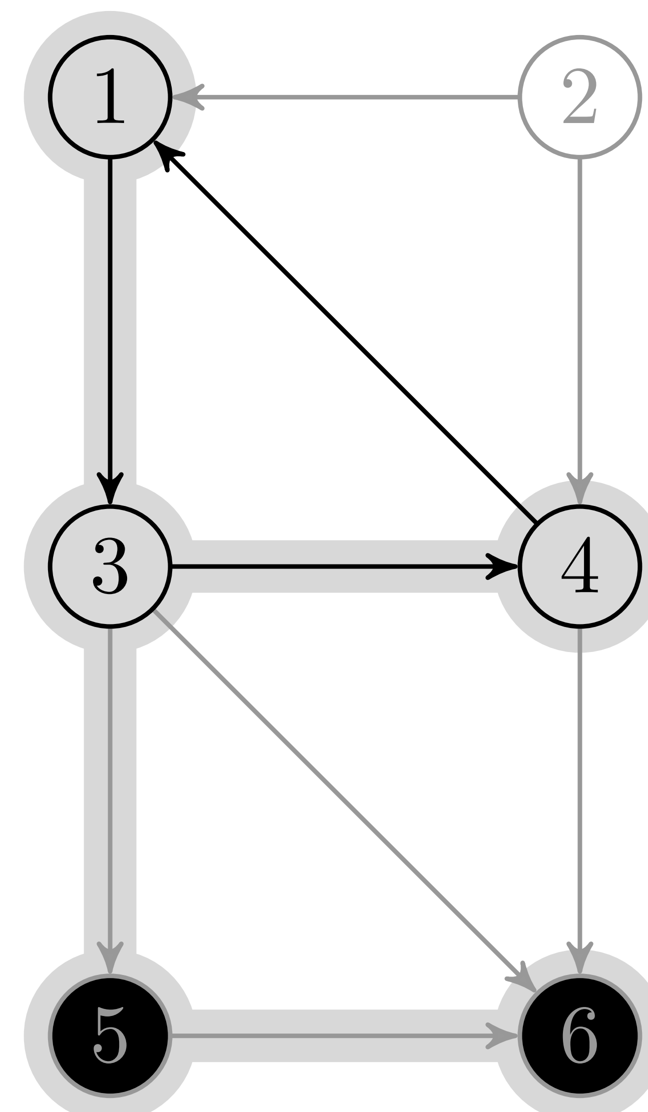


Er v grå? Bakoverkant

DFS-VISIT(G, u)

```
1   time = time + 1
2   u.d = time
3   u.color = GRAY
4   for each v ∈ G.Adj[u]
5       if v.color == WHITE
6           v.π = u
7           DFS-VISIT(G, v)
8   u.color = BLACK
9   time = time + 1
10  u.f = time
```

$u, v = 1, 3 \rangle 3, 4 \rangle 4, 1$



Kallstakk: Sykel

DFS-VISIT(G, u)

$time = time + 1$

$u.d = time$

$u.color =$ GRAY

for each $v \in G.Adj[u]$

  if $v.color ==$ WHITE

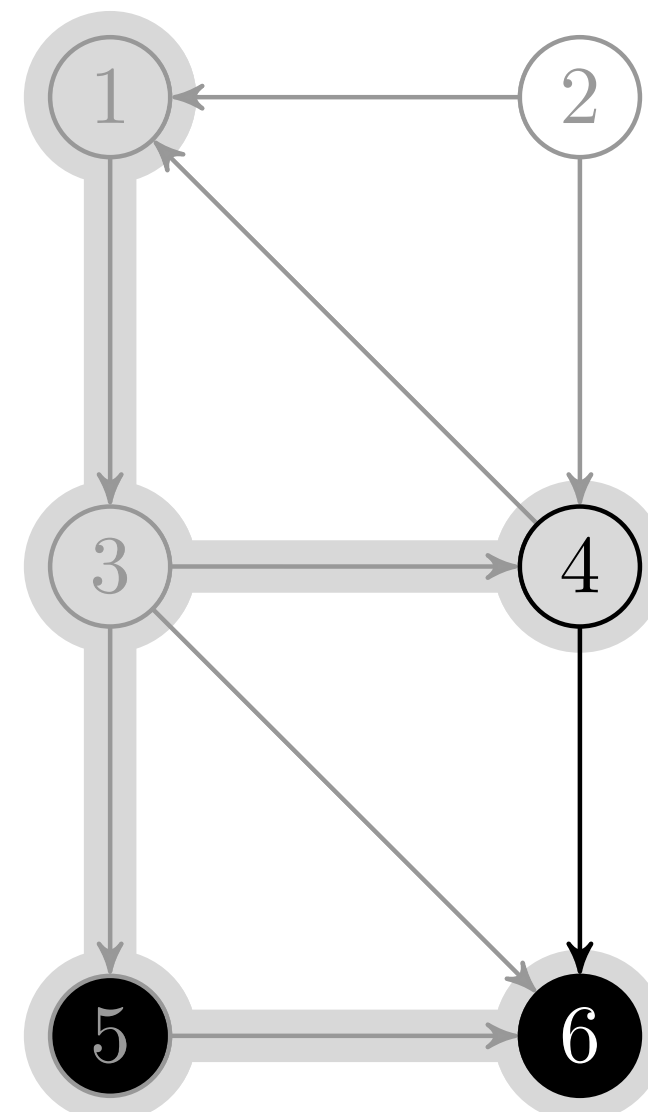
    $v.\pi = u$

    DFS-VISIT(G, v)

$u.color =$ BLACK

$time = time + 1$

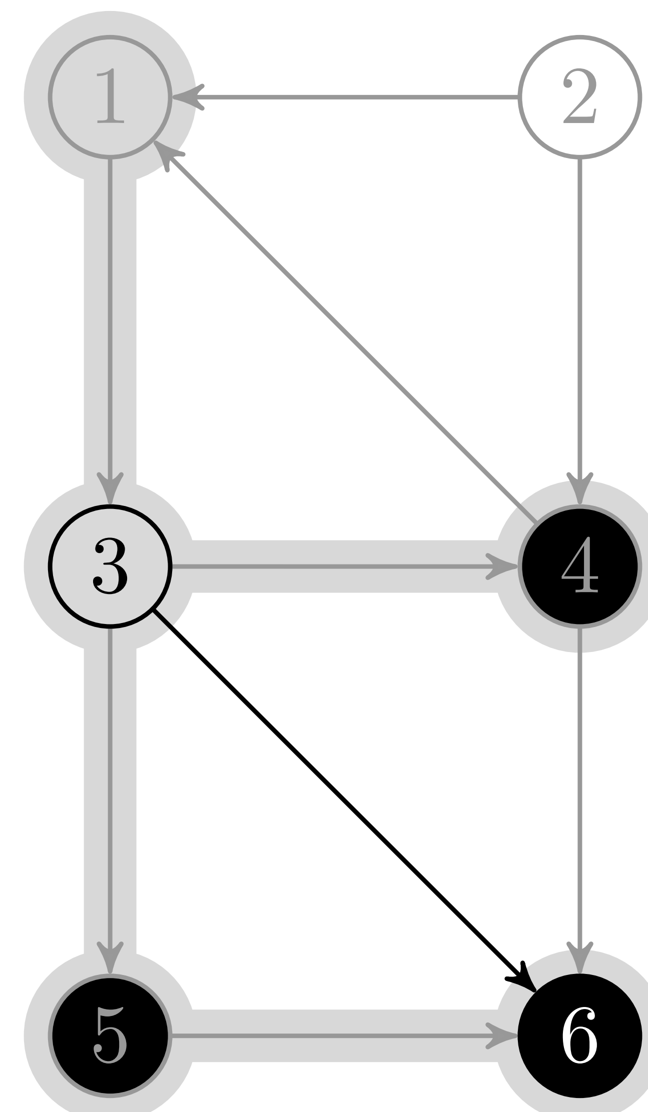
$u.f = time$



$u, v = 1, 3 \rangle 3, 4 \rangle 4, 6$

Svart $(v.d < u.d)$: Krysskant

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

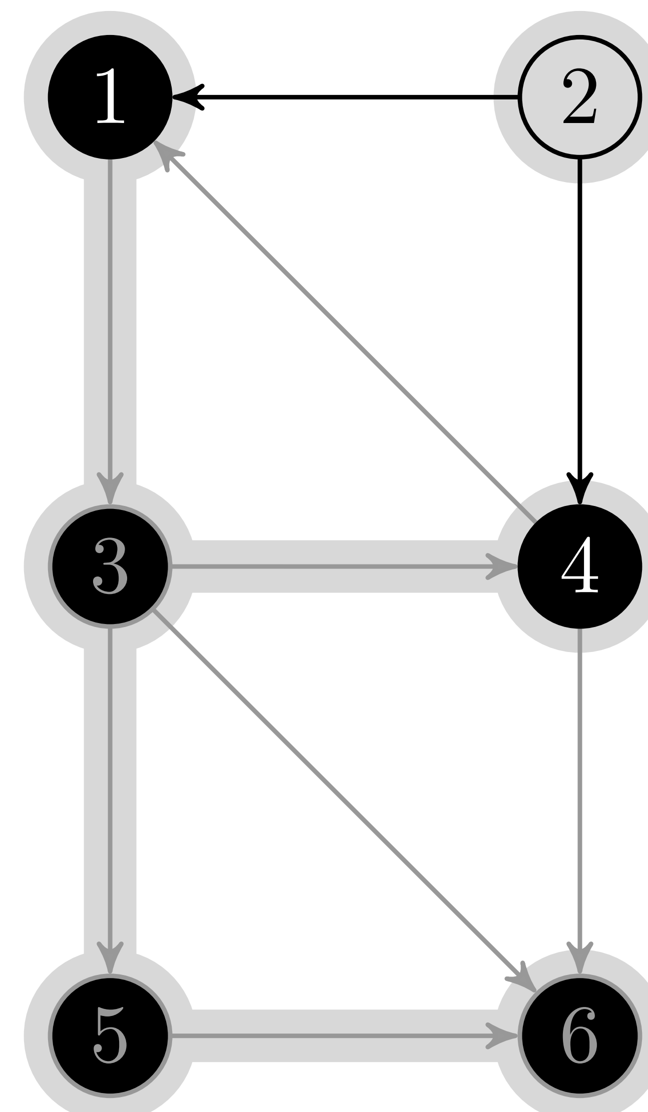


$u, v = 1, 3 \rangle 3, 6$

Svart $(v.d > u.d)$: Foroverkant

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

$u, v \;=\; 2, -$



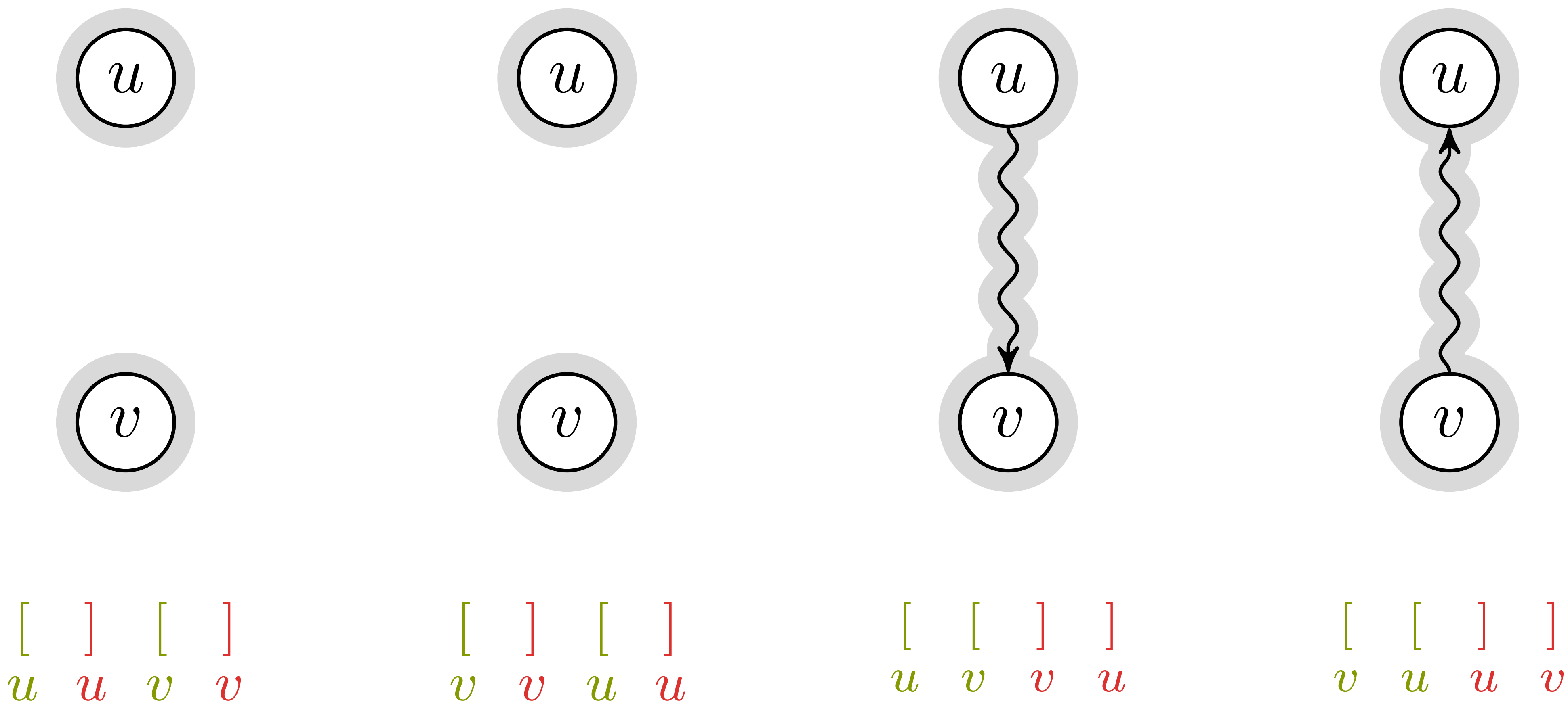
Flere krysskanter

Traversering › DFS ›

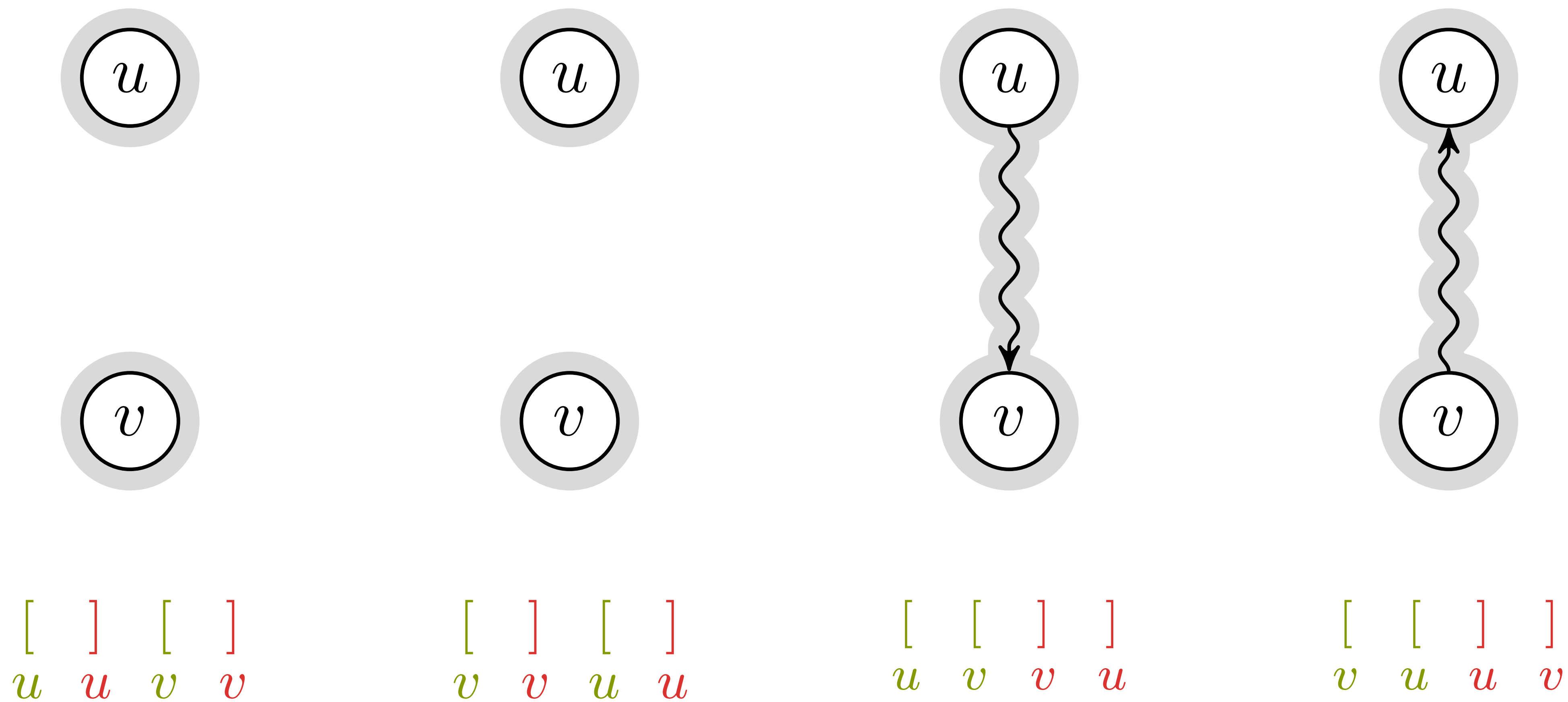
Parentesteoremet

Obs: Etterkommere i DFS-skogen!

Dvs., det kan godt hende det går stier mellom u og v i grafen, men at ingen av dem er etterkommer av den andre i DFS-skogen.



Noder oppdages før og avsluttes etter sine etterkommere



Dette er de eneste mulighetene!

Kjøretid

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

Alle noder farges grå og svarte; alle kanter undersøkes

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!

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)

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 <u>ikke</u> fra alle i vår versjon (men kunne ha gjort det)

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

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!

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): $\Theta(V \lg V + E \lg E) =$

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): $\Theta(V \lg V + E \lg E) = \Theta(E \lg V)$

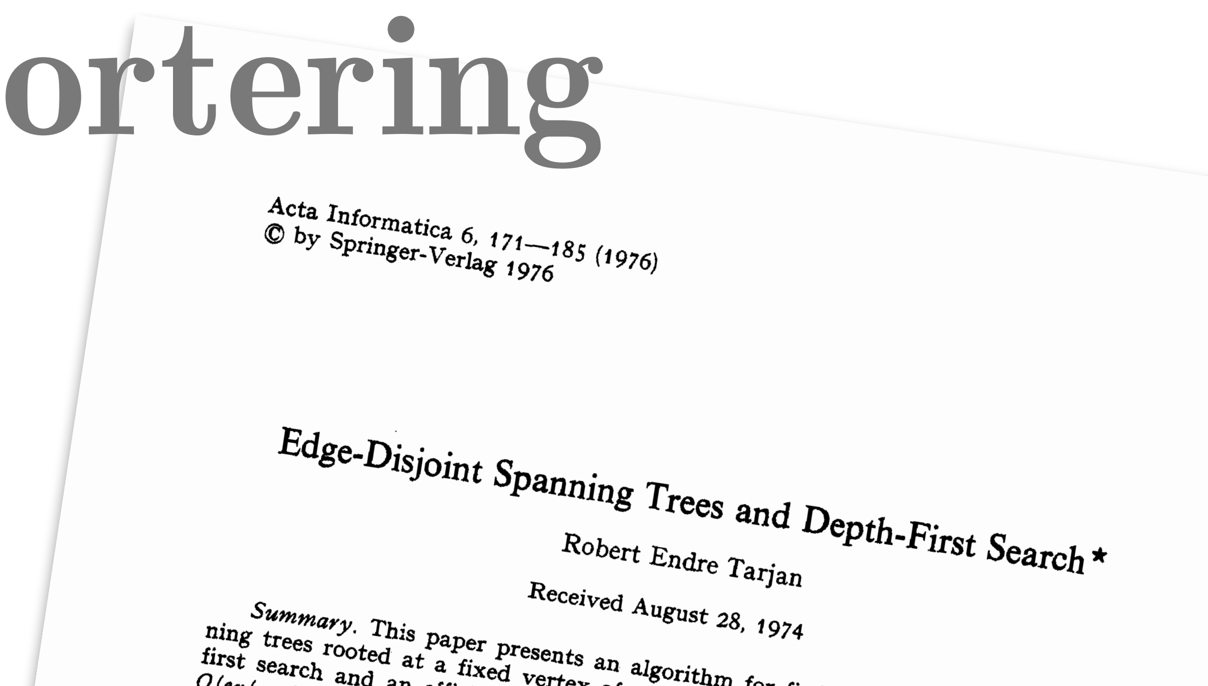
Algoritme	**WC**	**BC**
Dybde-først-søk	$\Theta(V + E)$	$\Theta(V + E)$
Bredde-først-søk	$\Theta(V + E)$	$\Theta(V)$
______-først-søk	$\Omega(V + E)$	$\Theta(V)$

Bortsett fra i DFS starter vi vanligvis kun ett sted

4:4

Topologisk sortering

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


Acta Informatica 6, 171—185 (1976)

Edge-Disjoint Spanning Trees and Depth-First Search*

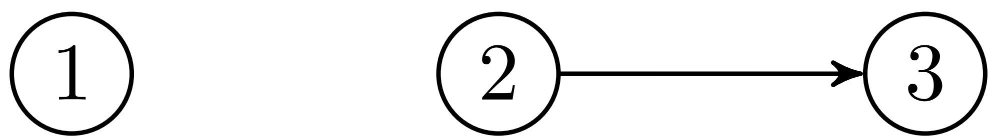
Robert Endre Tarjan

Received August 28, 1974

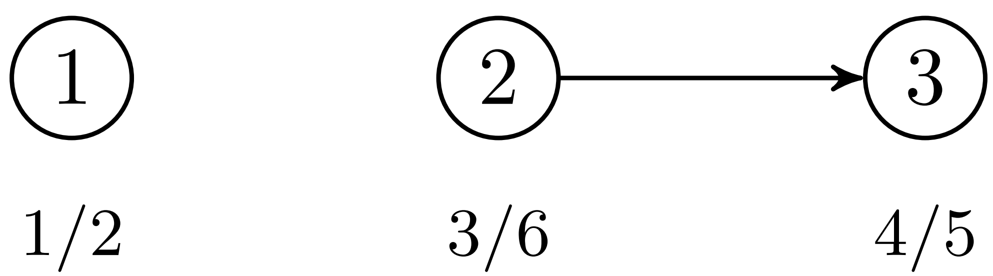
Summary. This paper presents an algorithm for
ning trees rooted at a fixed vertex of
first search and an

Topologisk sortering

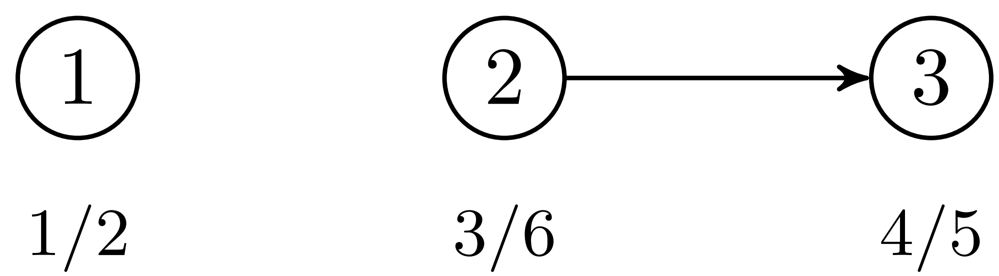
- Gir nodene en rekkefølge
- Foreldre før barn
- Evt.: Alle kommer etter avhengigheter
- Det er egentlig det vi gjør med delproblemgrafen i dynamisk programmering
- Krever at grafen er en DAG!



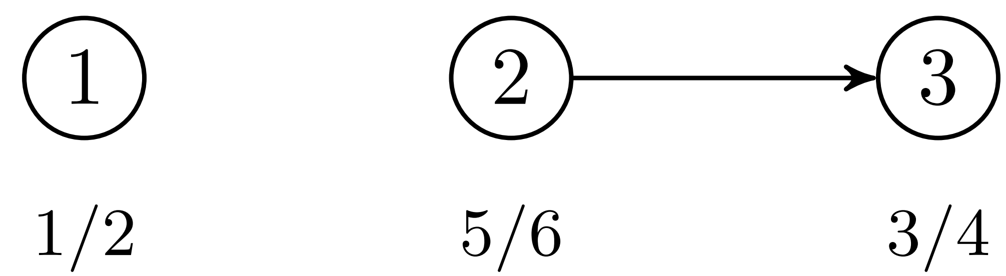
Her må 2 komme før 3; ellers fritt frem



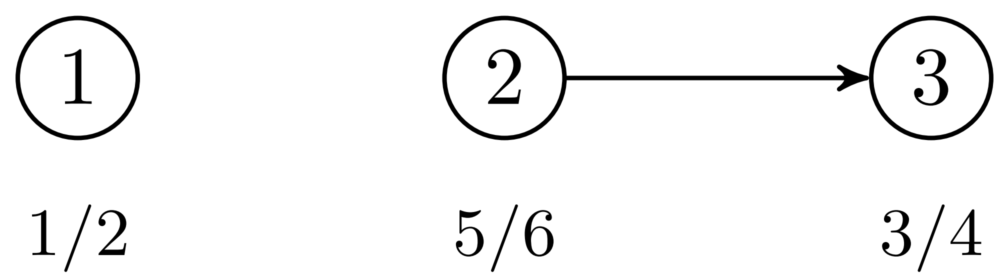
Mulige *discover-/finish-times*



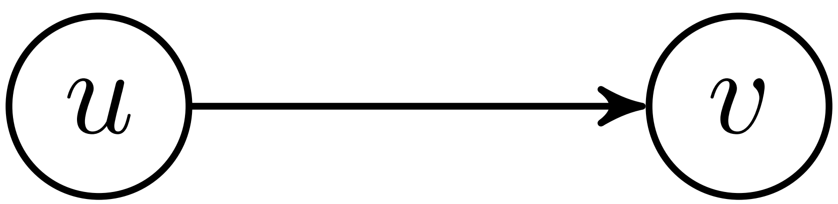
Her oppdages 2 før 3, men det er ikke garantert!



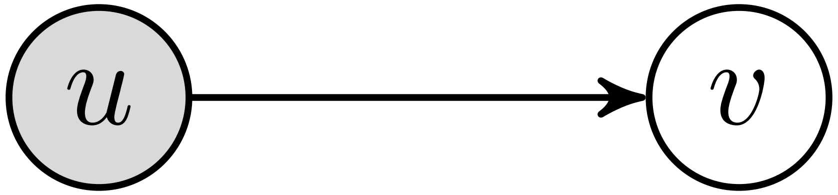
Her kalles DFS-VISIT(G, 3) før DFS-VISIT(G, 2)



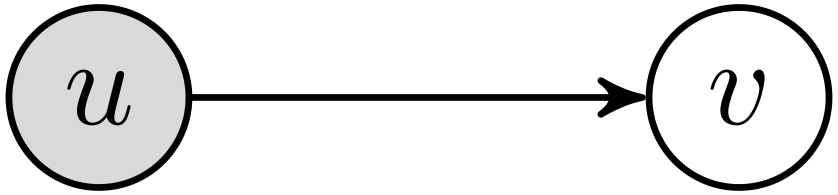
Merk at 2 fortsatt har høyere finish-time enn 3!



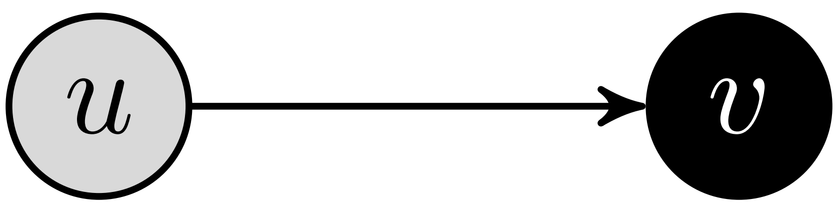
La oss si vi undersøker kanten (u, v) under DFS



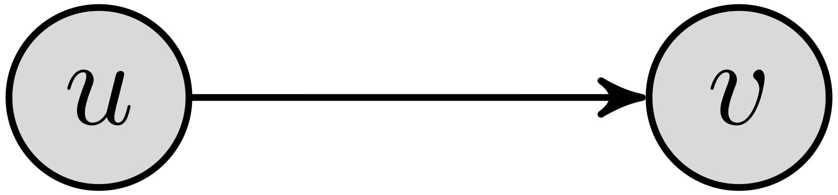
La oss si vi undersøker kanten (u, v) under DFS



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



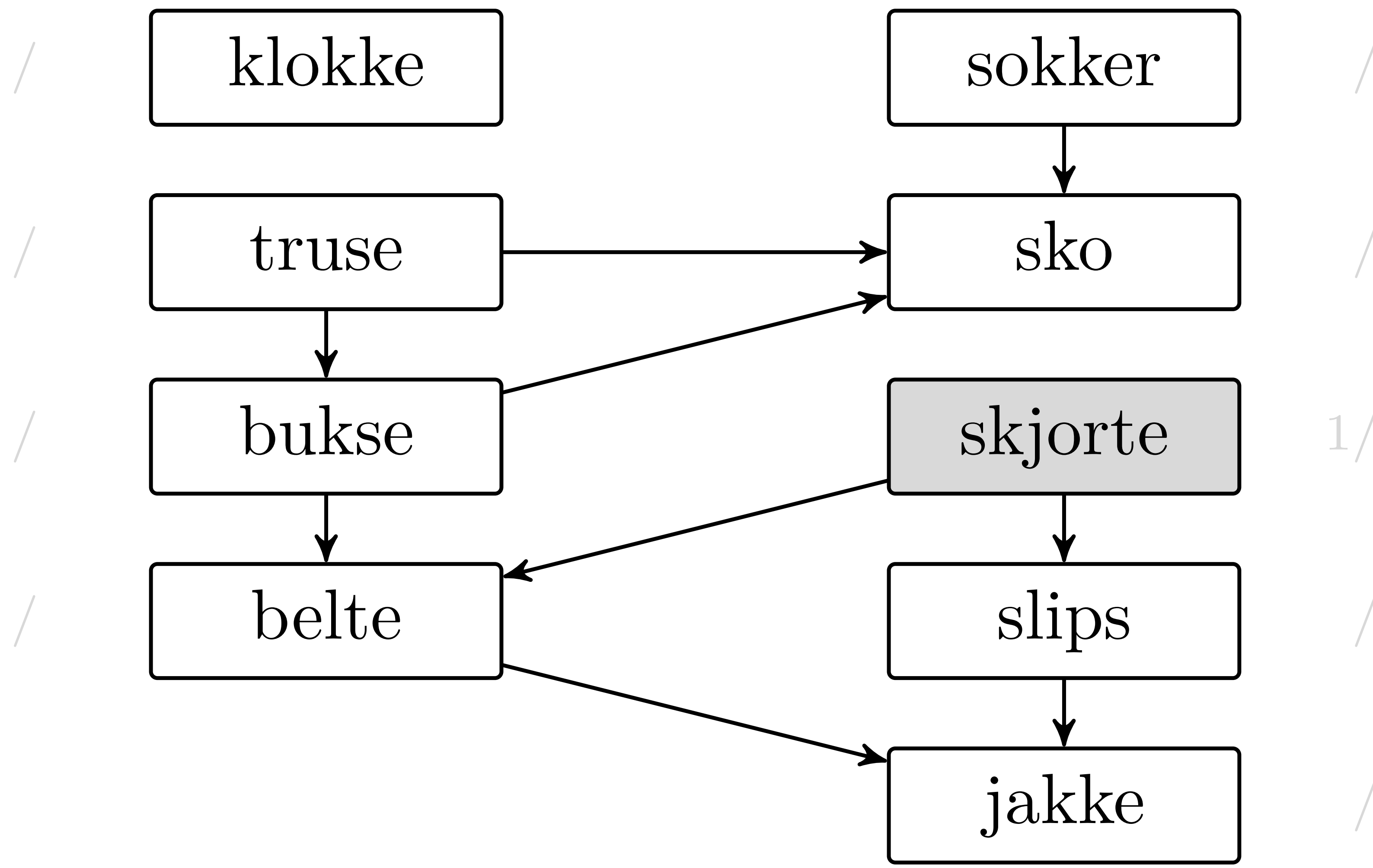
Hvis v er svart, så er den alt ferdig: $u.f > v.f$

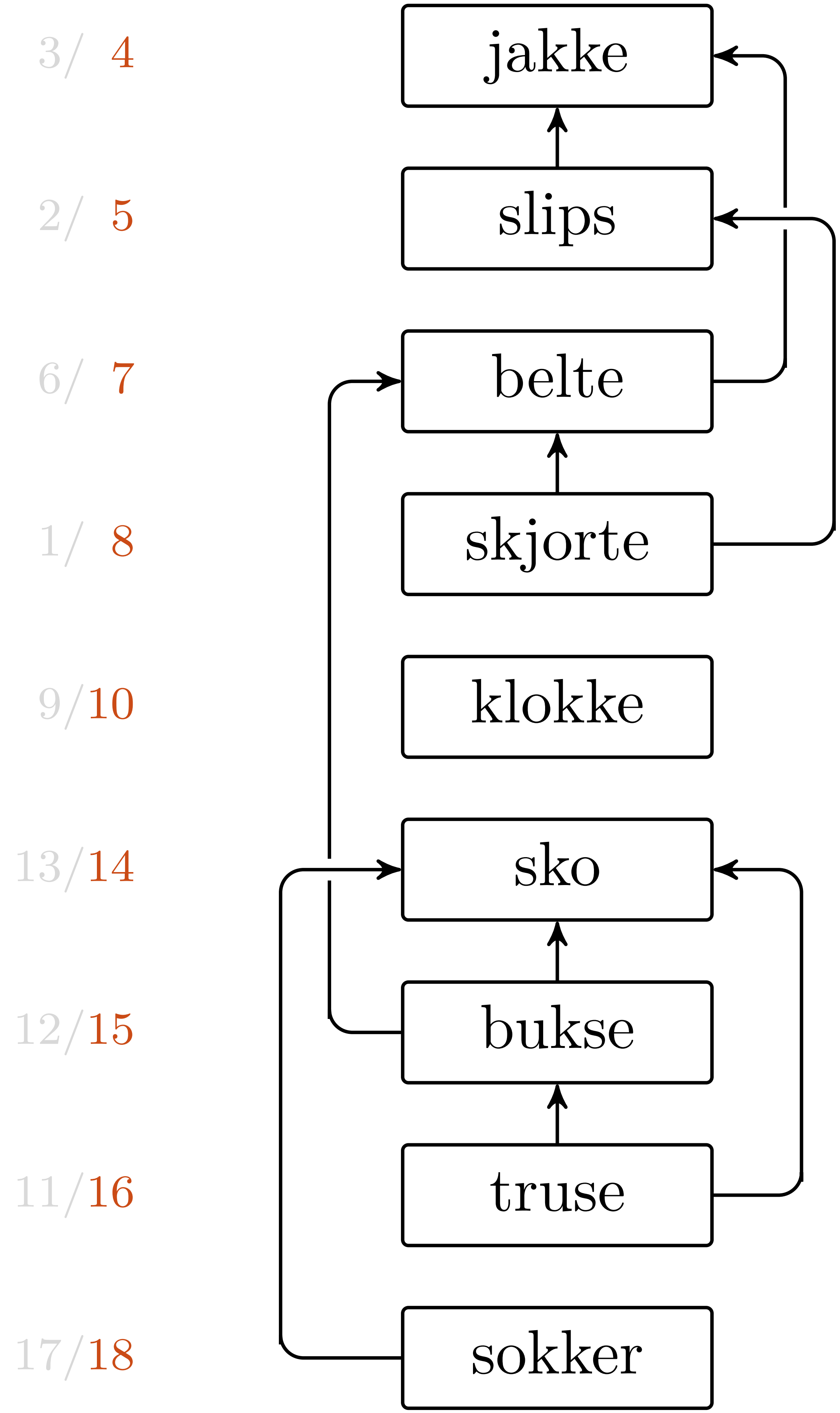


Hvis v er grå, så har vi en sykel – og det er umulig!

Altså ...

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


klokke
sokker
truse
sko
bukse
skjorte
1
belte
slips
jakke


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

- **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?

1. Grafrepresentasjoner

2. Bredde-først-søk

3. Dybde-først-søk

4. Topologisk sortering