Forelesning 8

Traversering av grafer

Repetisjon og oppklaring

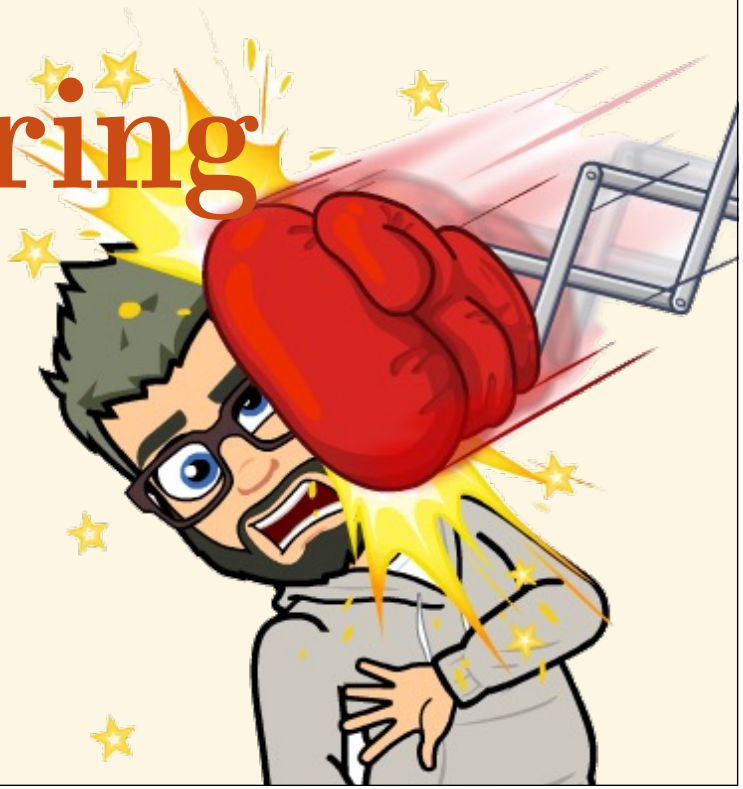
- Ryggsekkproblemet (0-1)
- Huffman, korrekthet
- Velfundert induksjon

Traversering

- Bredde-først-søk
- Iterativt dybde-først-søk
- Dybde-først-søk
- Topologisk sortering

Fra tidligere ...

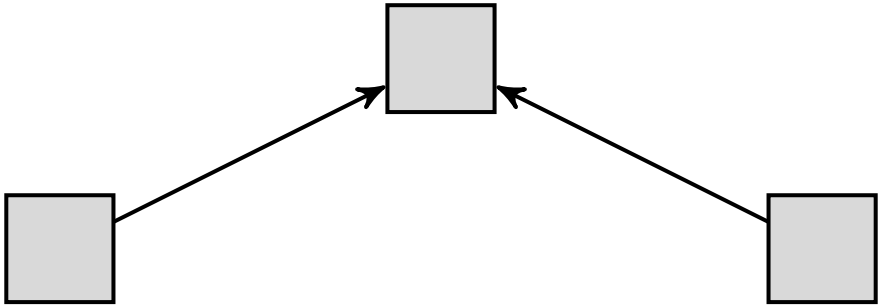
Dynamisk programmering

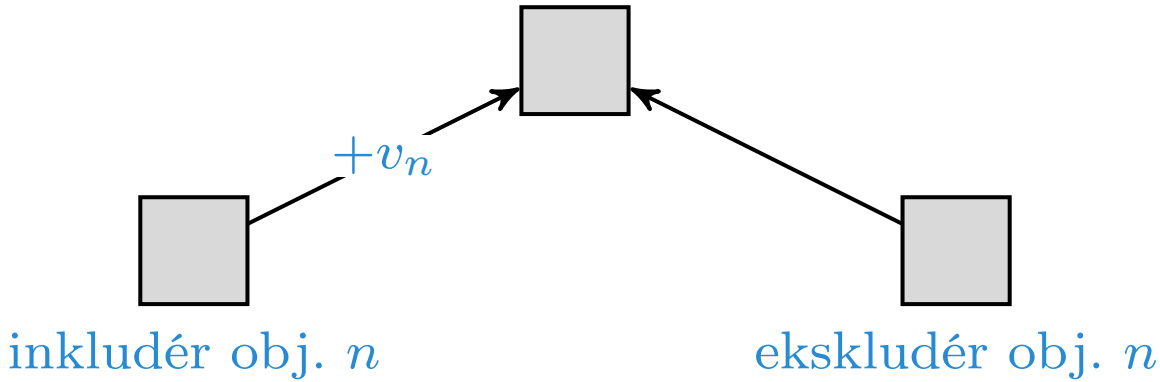


DP › Ryggsekk

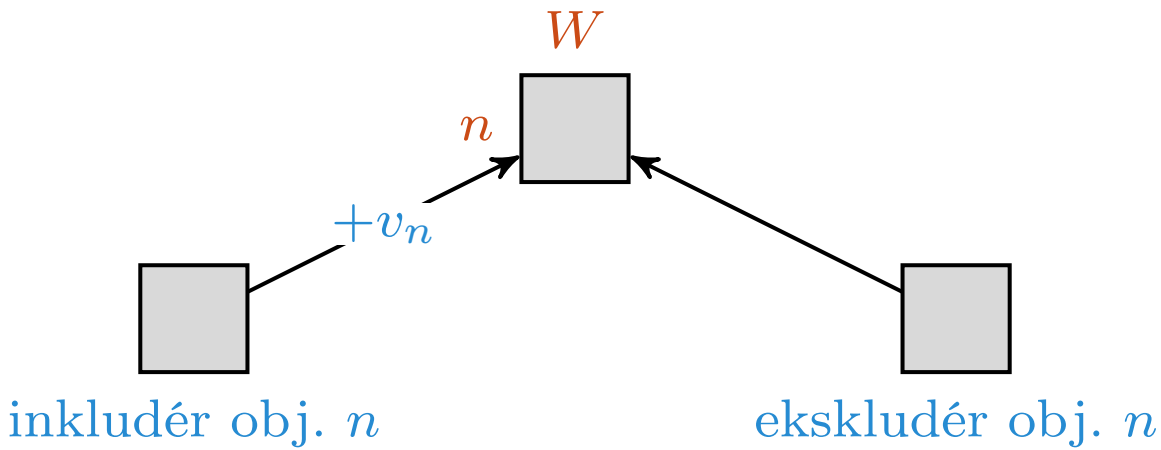
Input: Verdier $v_1, \ldots, v_n$, vekter $w_1, \ldots, w_n$ og en kapasitet W.

Output: Indekser $i_1, \ldots, i_k$ slik at $w_{i_1} + \cdots + w_{i_k} \leq W$ og totalverdien $v_{i_1} + \cdots + v_{i_k}$ er ma[illegible]

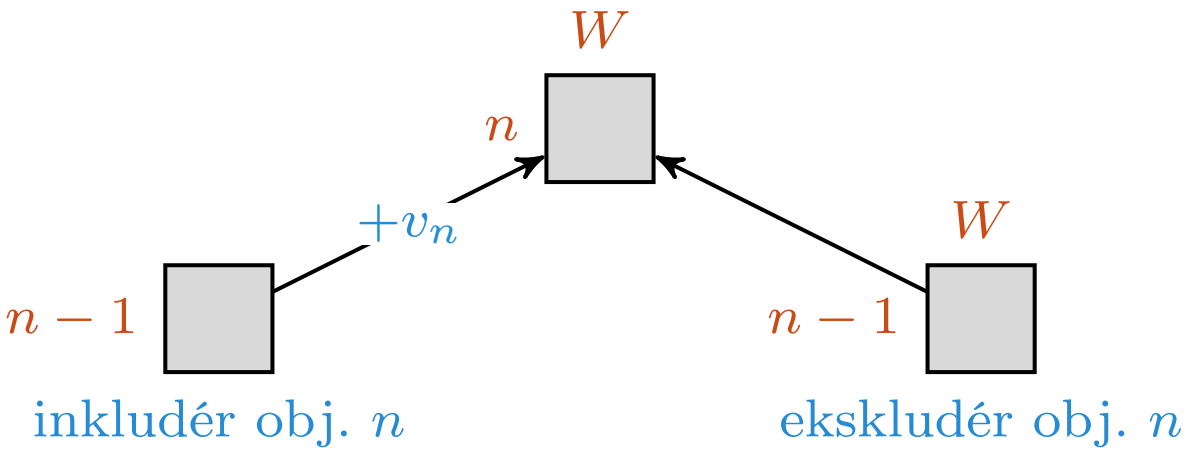




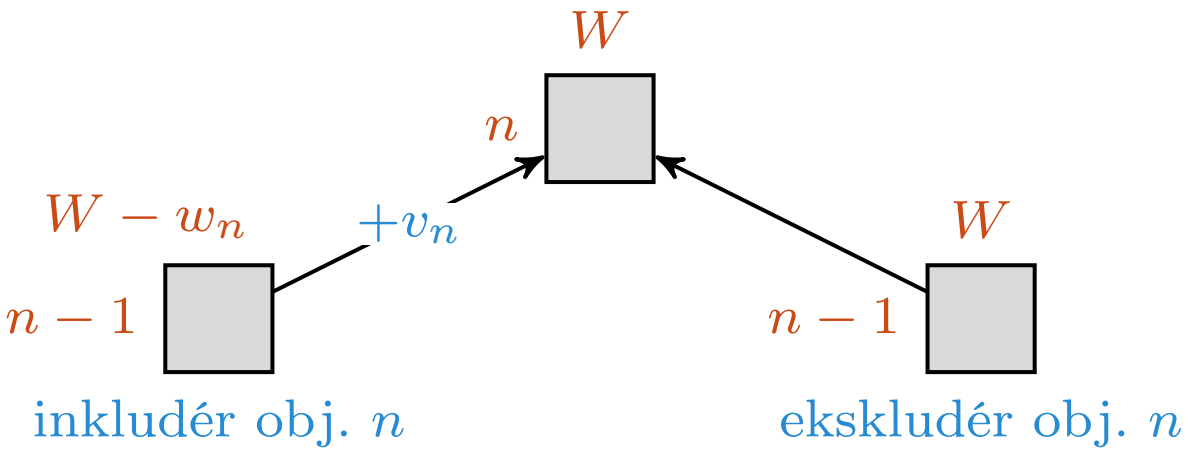
Objekt n bidrar med verdi v_n



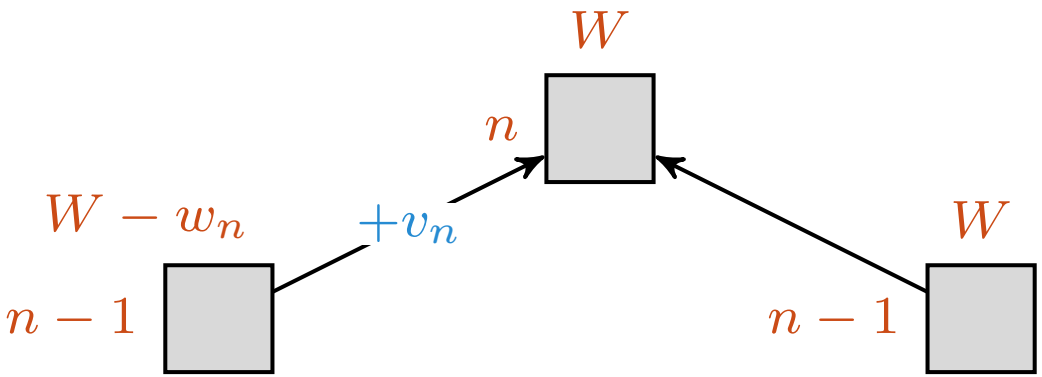
Vi parametriserer delproblemer vha. n og W

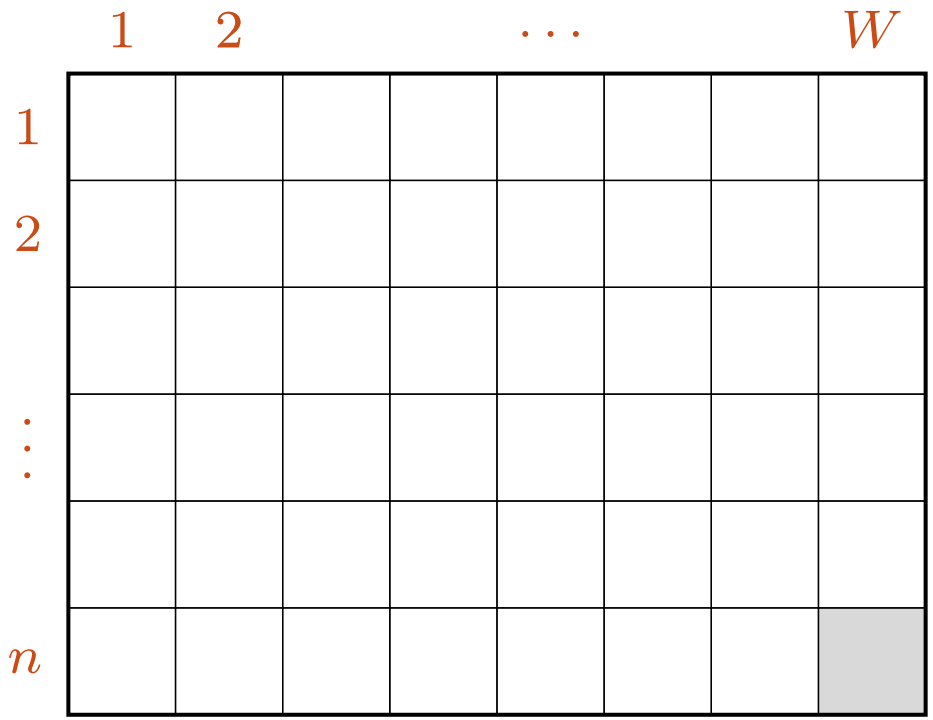


Ser nå bare på objekter $1 \ldots n-1$

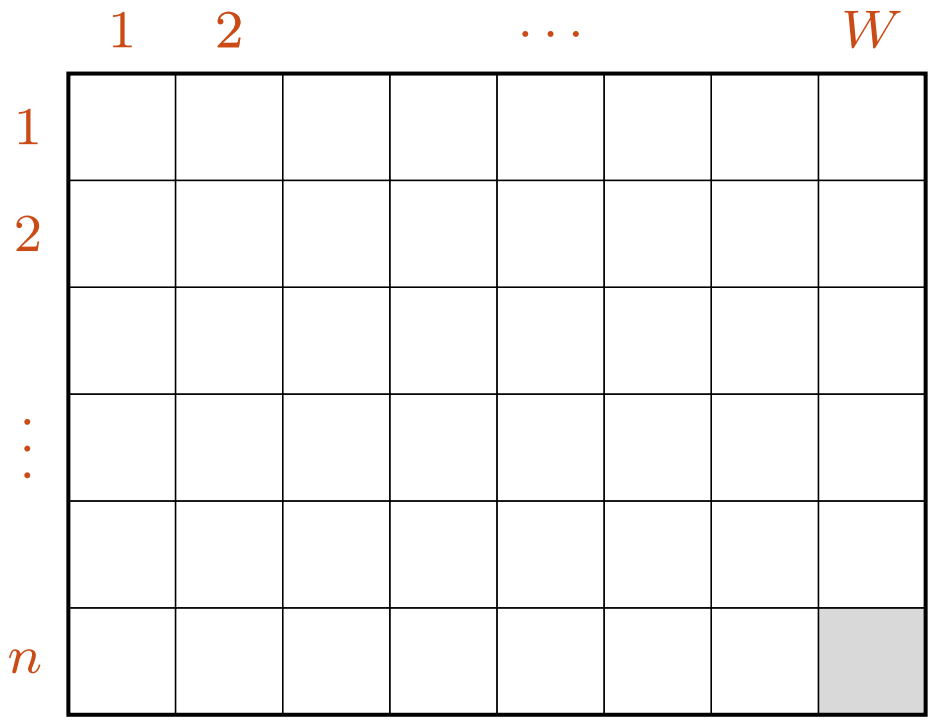


Objekt n bruker opp w_n av W

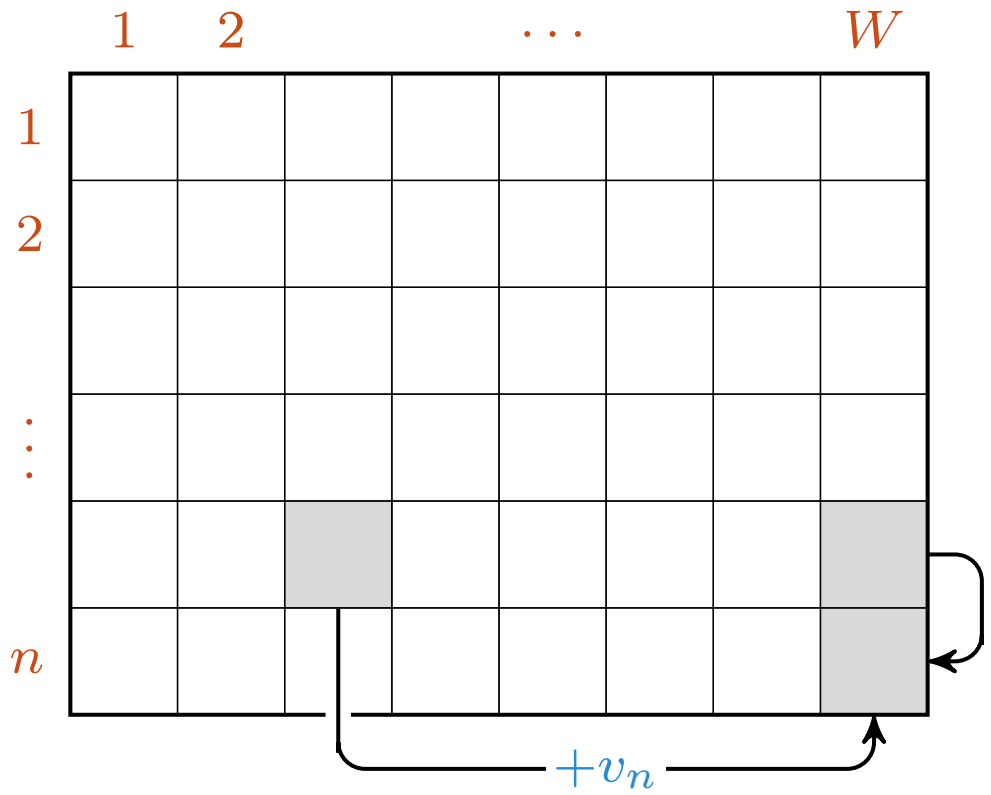

W
n
$W - w_n$
$+v_n$
W
$n-1$
$n-1$



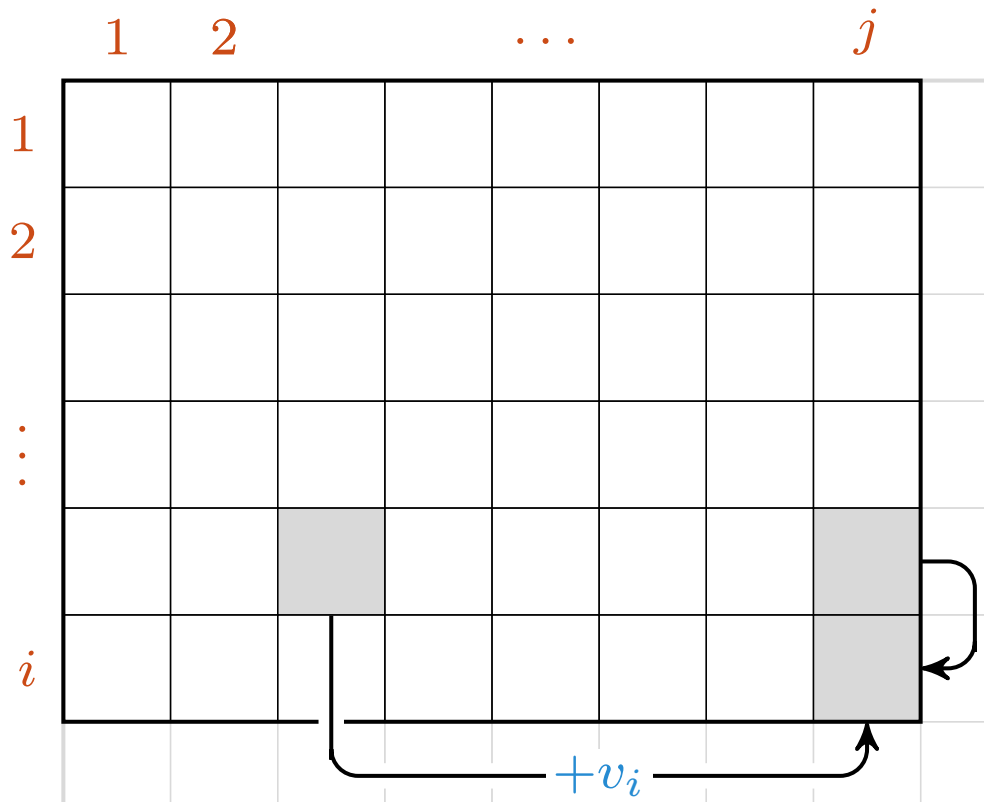
Lagre delløsninger i $n \times W$-tabell



La f.eks. $w_n = 5$.



Dekomponering som før; kan løses radvis

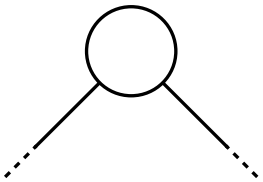


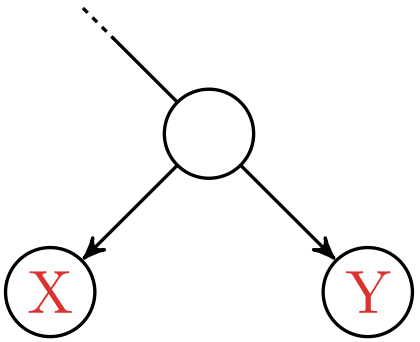
Samme dekomponering for alle delproblemer

```
ZERO-ONE-KNAPSACK(w, W, n)
1  let A[0..n, 0..W] be a new array
2  for j = 1 to W
3        A[0, j] = 0
4  for i = 1 to n
5        for j = 1 to W
6              A[i, j] = A[i − 1, j]
7              if w[i] ≤ j
8                    A[i, j] = max(A[i, j], A[i − 1, j − w] + v[i])
```

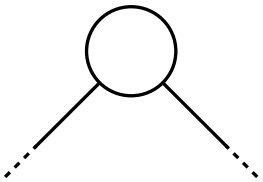
Grådighet

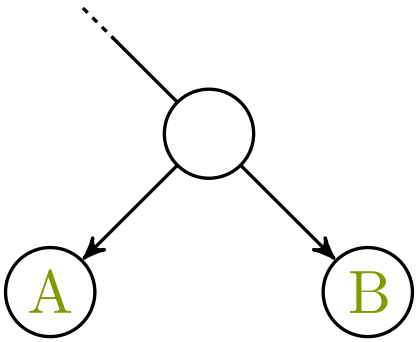
Korrekthet



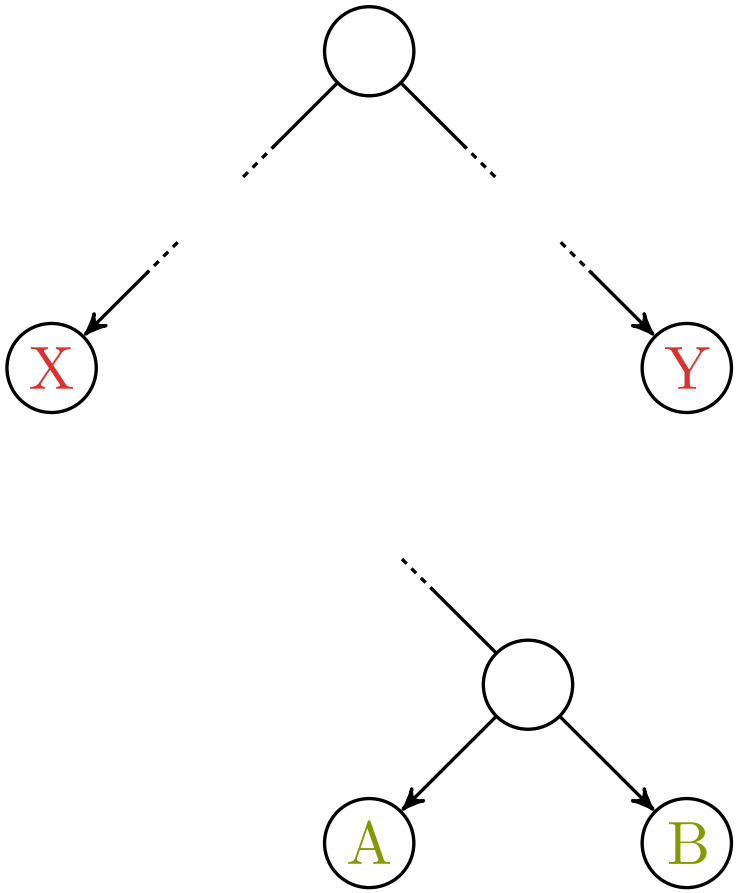


Vil vise: Sjeldneste sammen nederst lønner seg

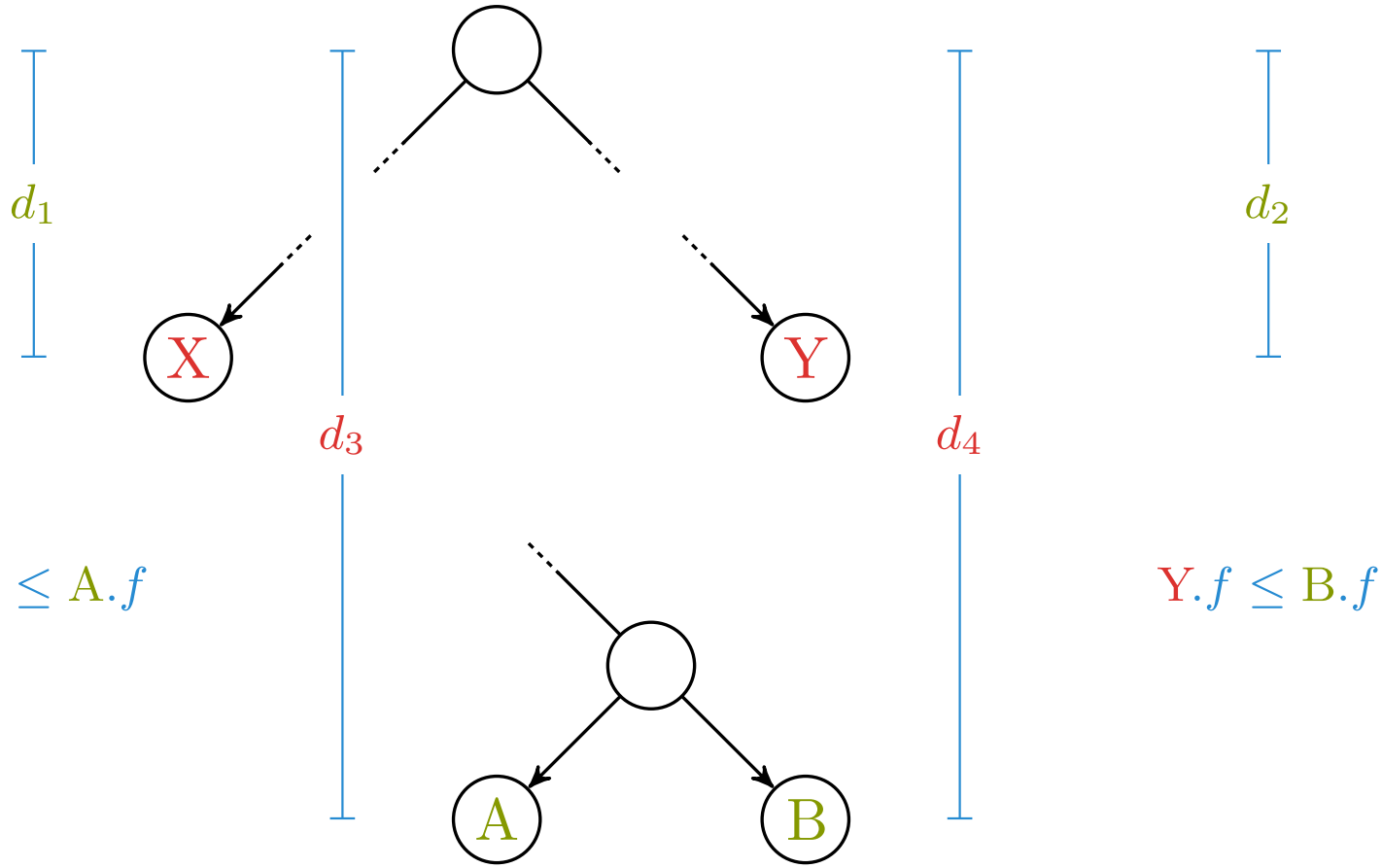




Anta en vilkårlig annen løsning . . .



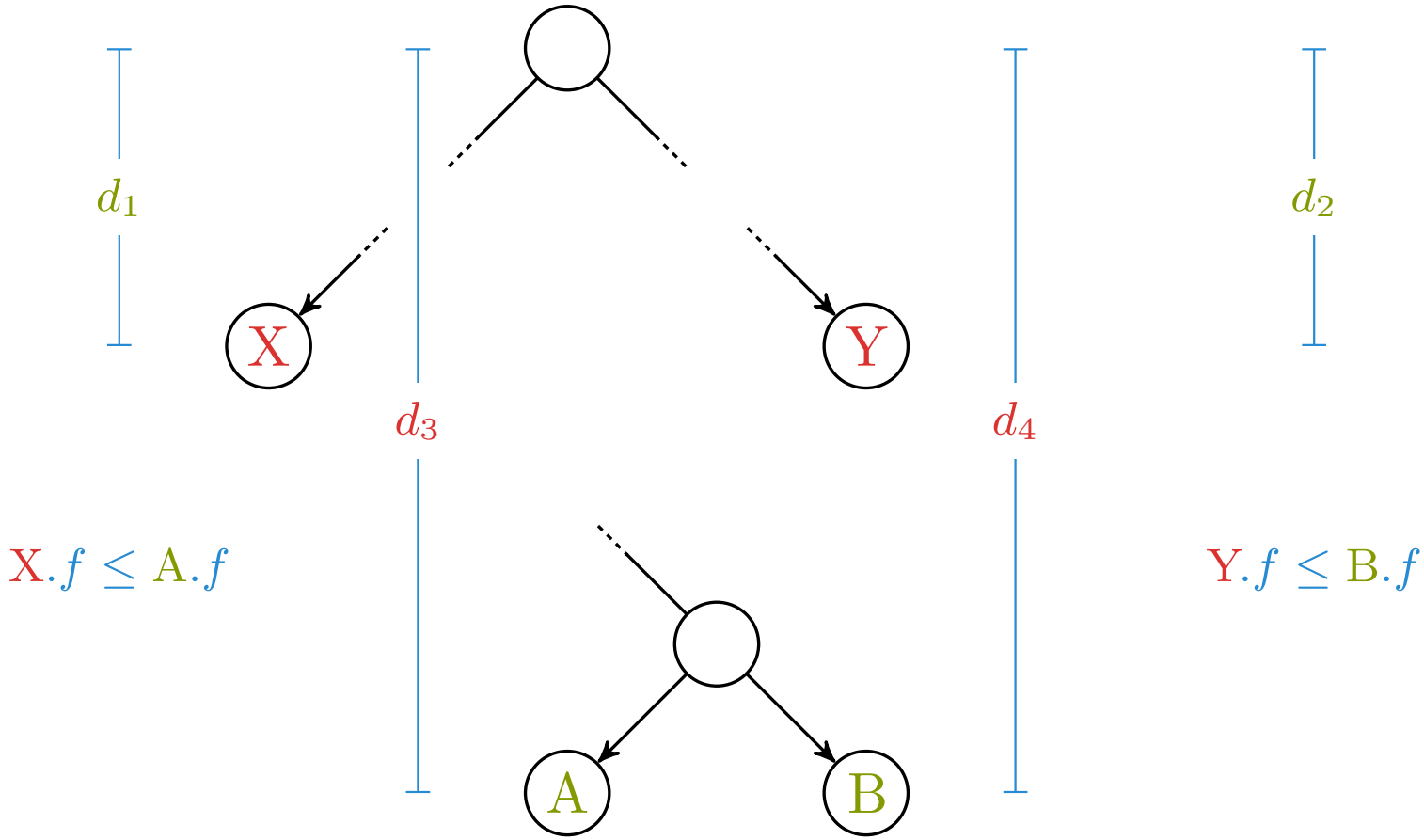
. . . med de sjeldneste lenger opp



$\mathrm{X}.f \leq \mathrm{A}.f$

$\mathrm{Y}.f \leq \mathrm{B}.f$

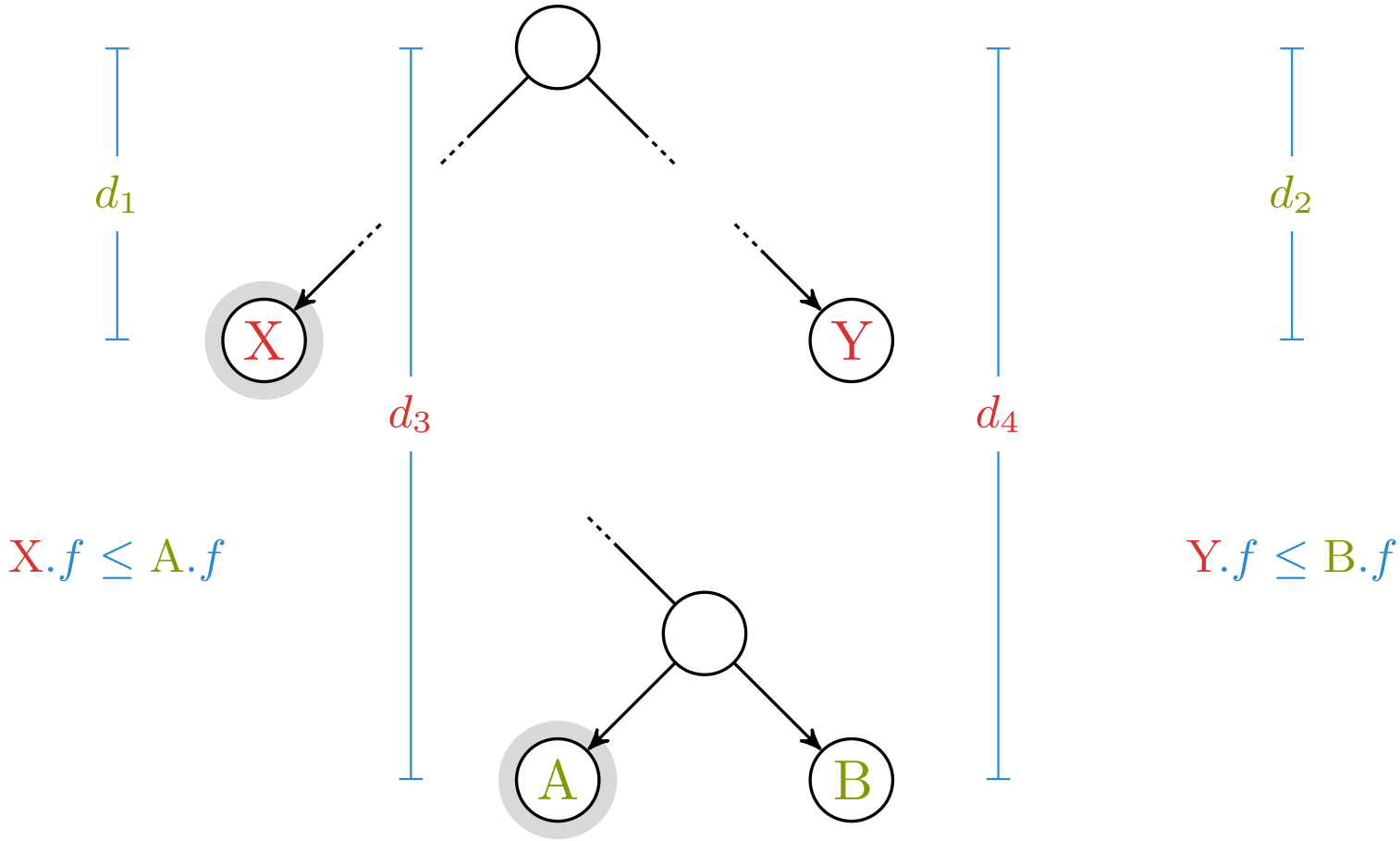
. . . med de sjeldneste lenger opp



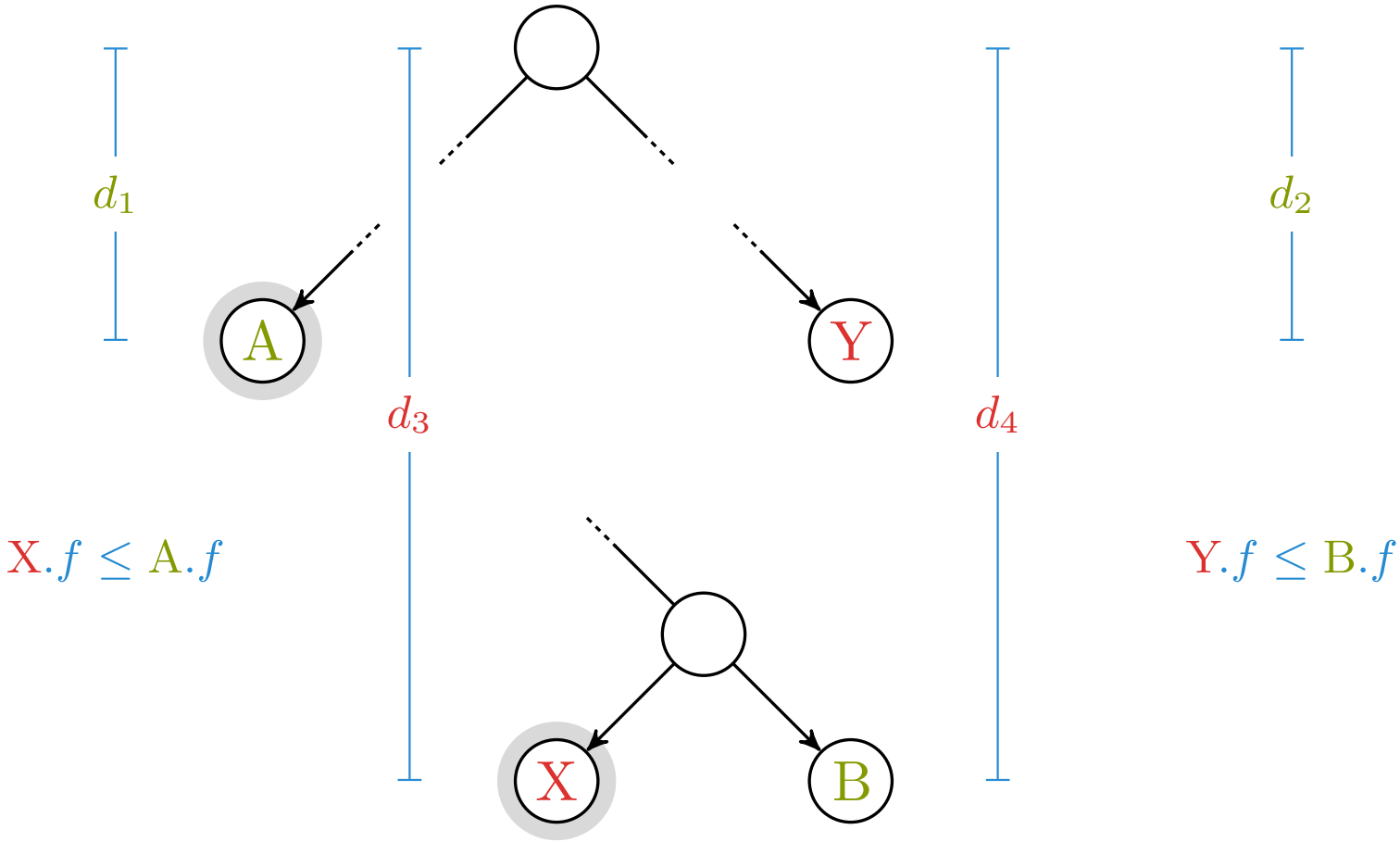
$\mathrm{X}.f \leq \mathrm{A}.f$

$\mathrm{Y}.f \leq \mathrm{B}.f$

Bidrag fra $\boxed{\mathrm{X}, \mathrm{Y}, \mathrm{A}, \mathrm{B}}$ er $\boxed{d_1\mathrm{X}.f + d_2\mathrm{Y}.f + d_3\mathrm{A}.f + d_4\mathrm{B}.f}$



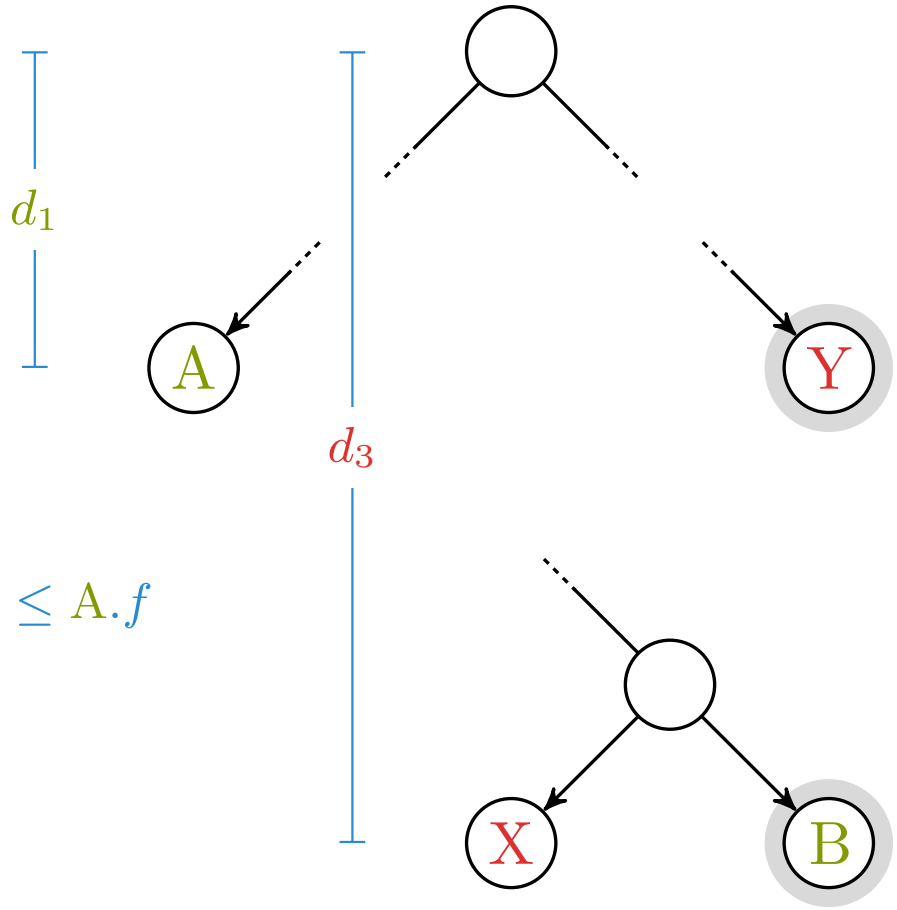
$$d_1\mathrm{X}.f + d_3\mathrm{A}.f$$



$\mathrm{X}.f \leq \mathrm{A}.f$

$\mathrm{Y}.f \leq \mathrm{B}.f$

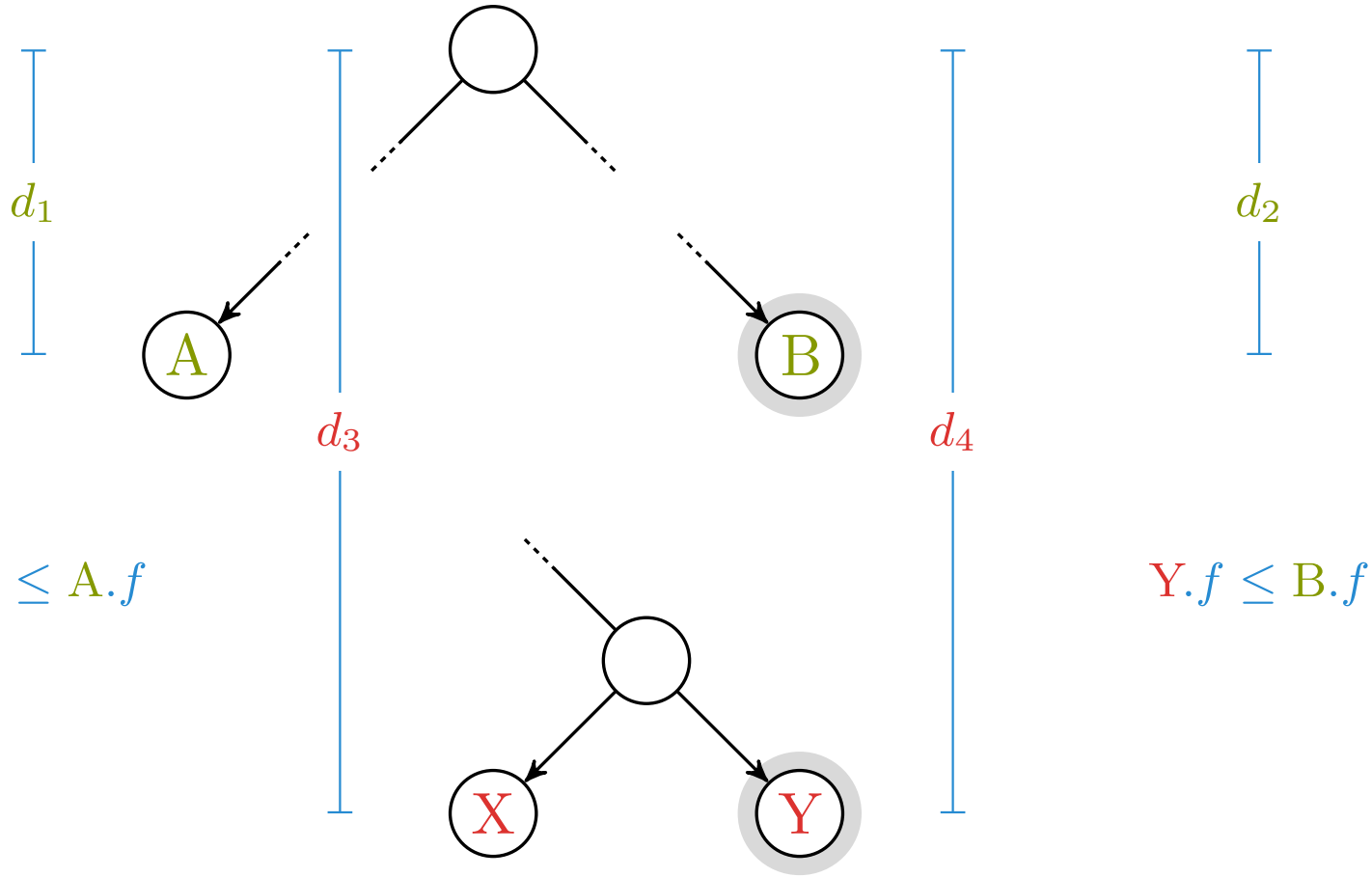
$$\boxed{d_1\mathrm{X}.f + d_3\mathrm{A}.f} \quad \geq \quad \boxed{d_1\mathrm{A}.f + d_3\mathrm{X}.f}$$



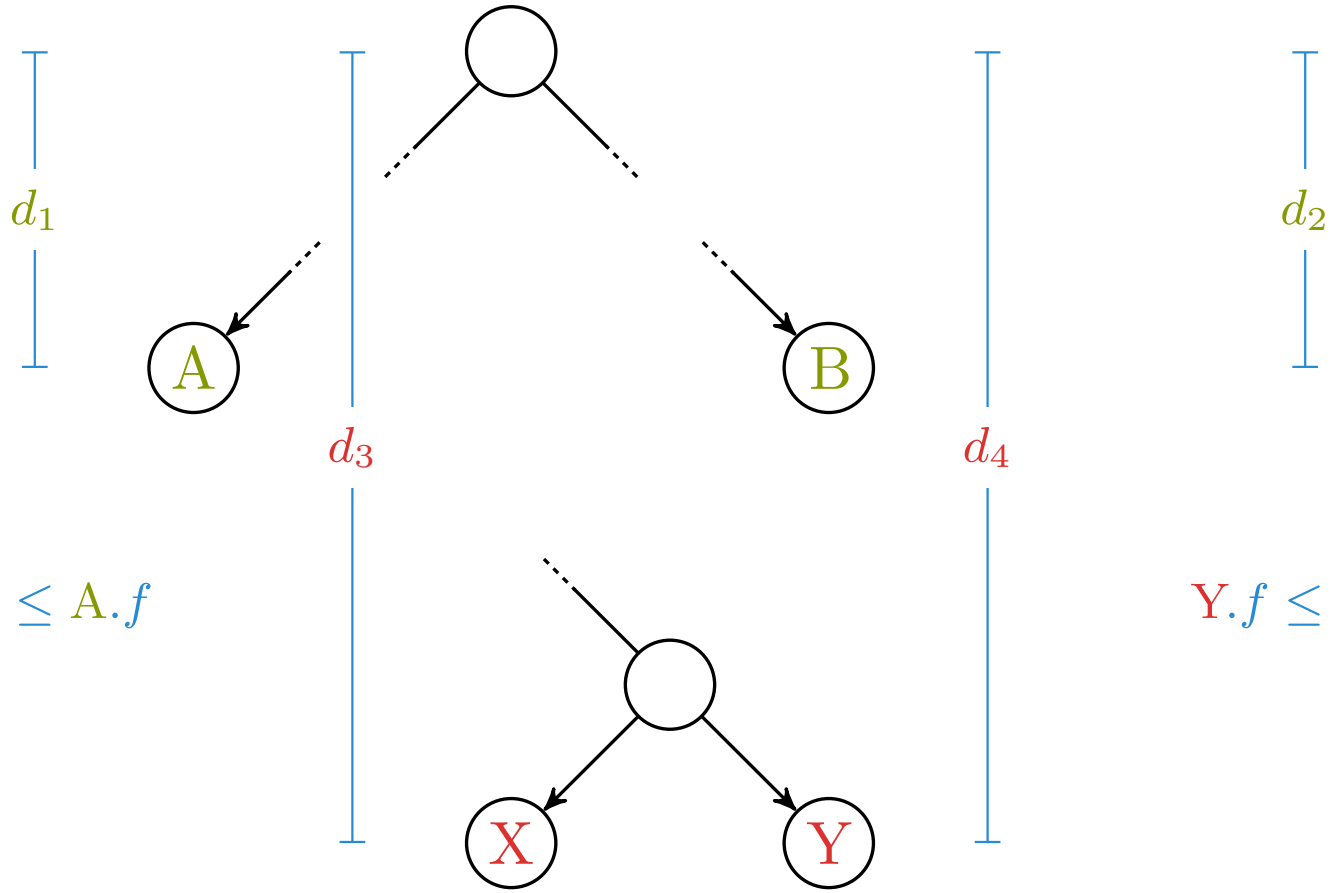
$\mathrm{X}.f \leq \mathrm{A}.f$

$\mathrm{Y}.f \leq \mathrm{B}.f$

$$d_2\mathrm{Y}.f + d_4\mathrm{B}.f$$



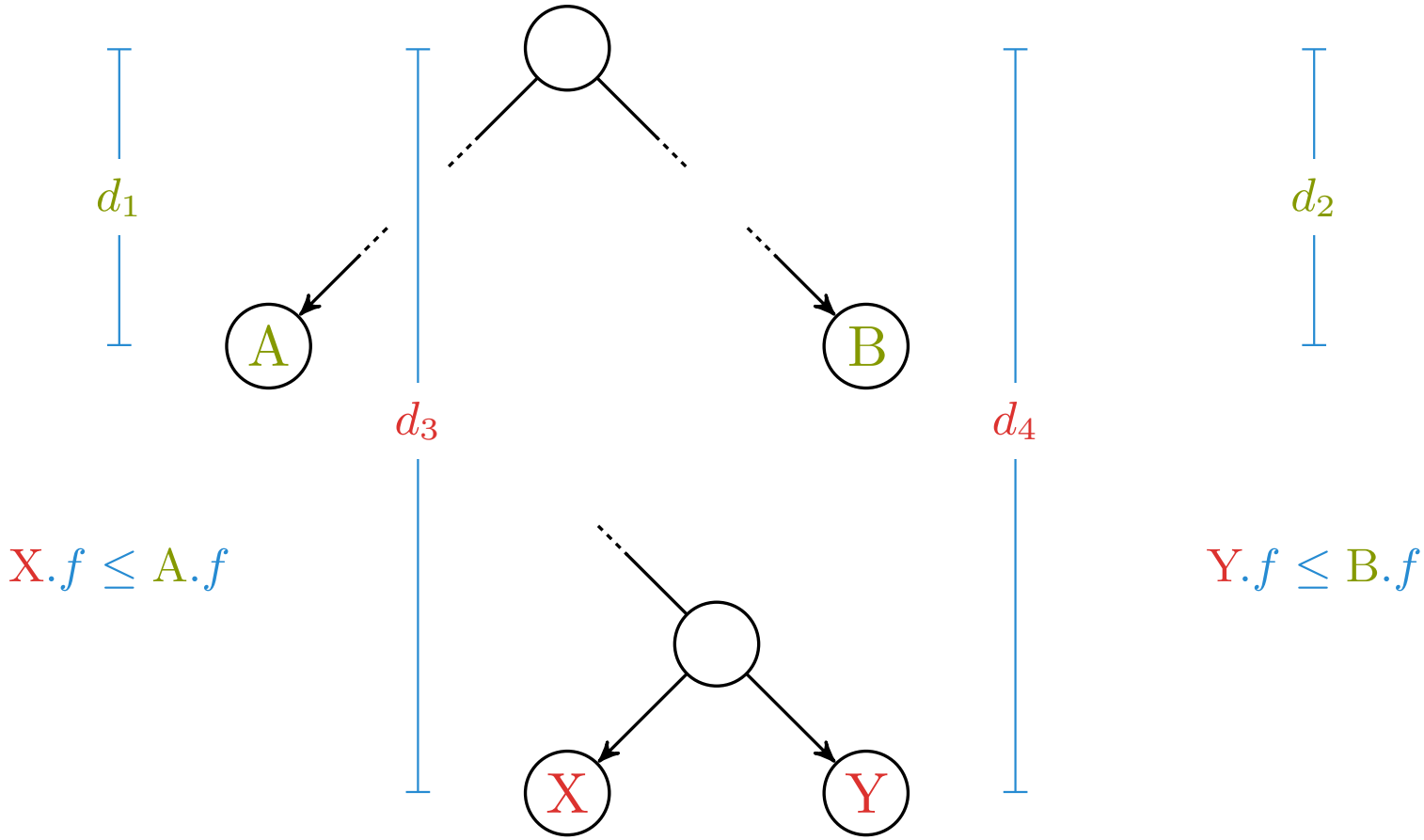
$$\boxed{d_2\mathrm{Y}.f + d_4\mathrm{B}.f} \quad \geq \quad \boxed{d_2\mathrm{B}.f + d_4\mathrm{Y}.f}$$



$\mathrm{X}.f \leq \mathrm{A}.f$

$\mathrm{Y}.f \leq \mathrm{B}.f$

$$d_1\mathrm{X}.f + d_2\mathrm{Y}.f + d_3\mathrm{A}.f + d_4\mathrm{B}.f \quad \geq \quad d_1\mathrm{A}.f + d_2\mathrm{B}.f + d_3\mathrm{X}.f + d_4\mathrm{Y}.f$$



Med andre ord: Vi taper ikke på å ha X og Y sammen nederst

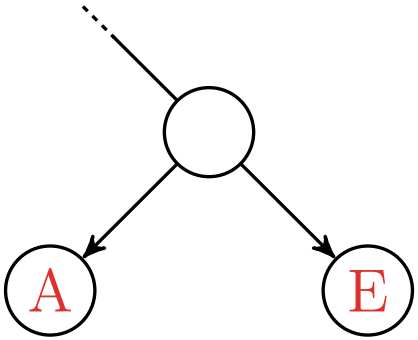
Greedy choice!

Altså: Å slå sammen X og Y er trygt som første trinn

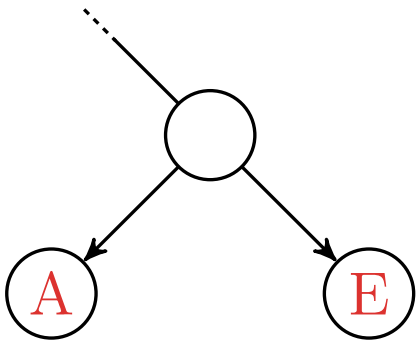
Optimal substruktur?

Men: Kan vi fortsette på samme måte?

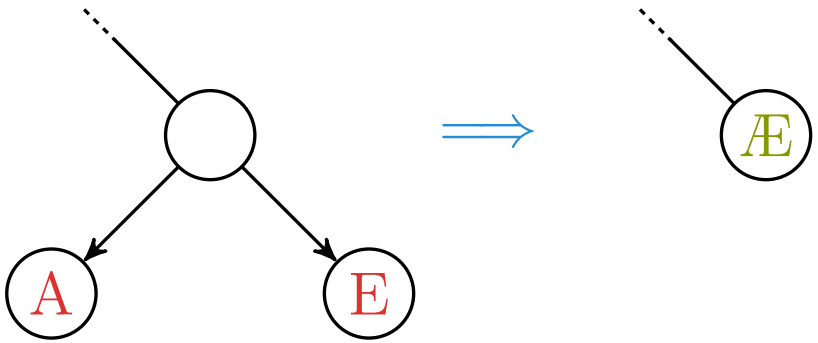
Det at det er trygt som et første trinn betyr ikke at vi bare kan fortsette sånn ... det må vi også bevise (optimal substruktur).



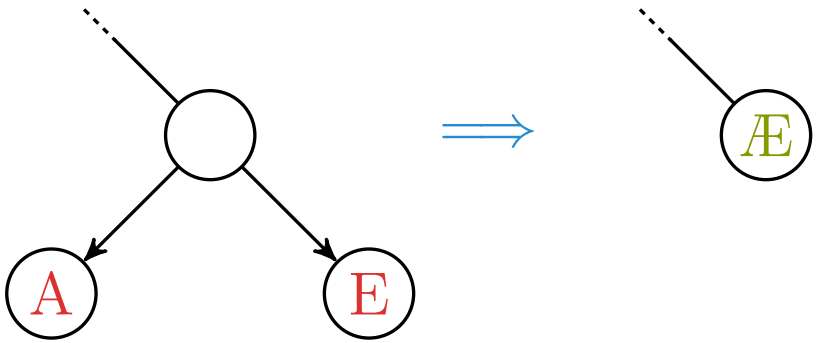
Bør resten bygges optimalt?



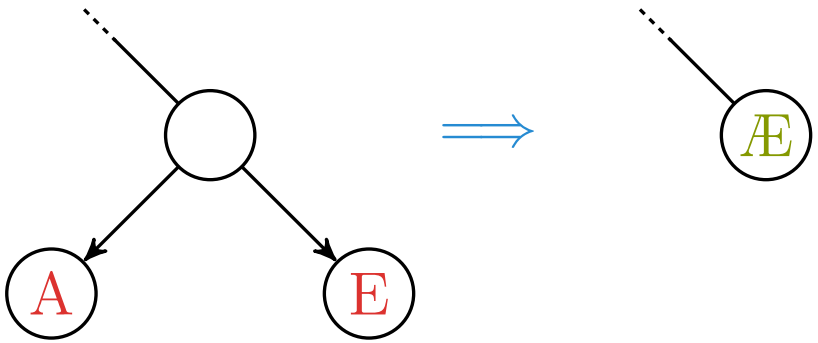
Konseptuelt: Behandle de to som ett tegn



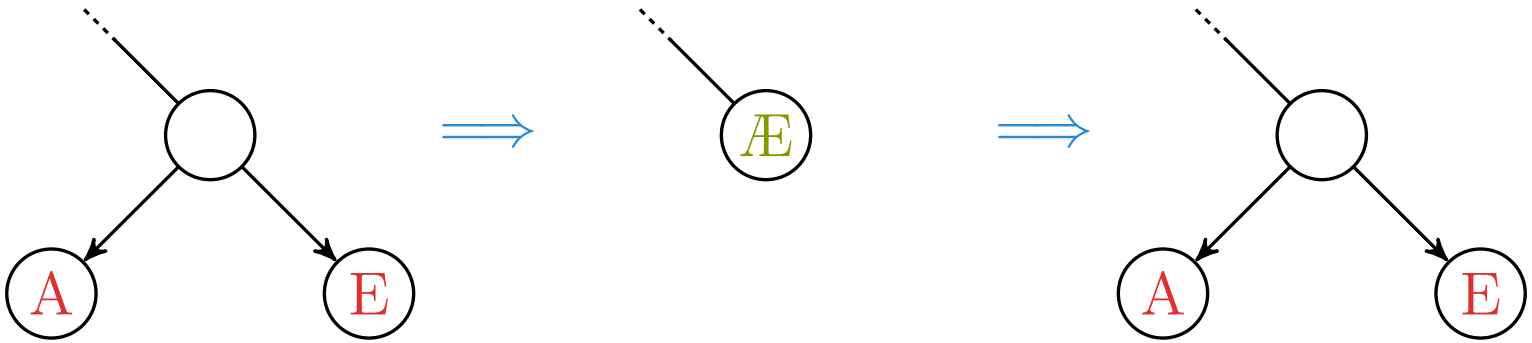
Konseptuelt: Behandle de to som ett tegn



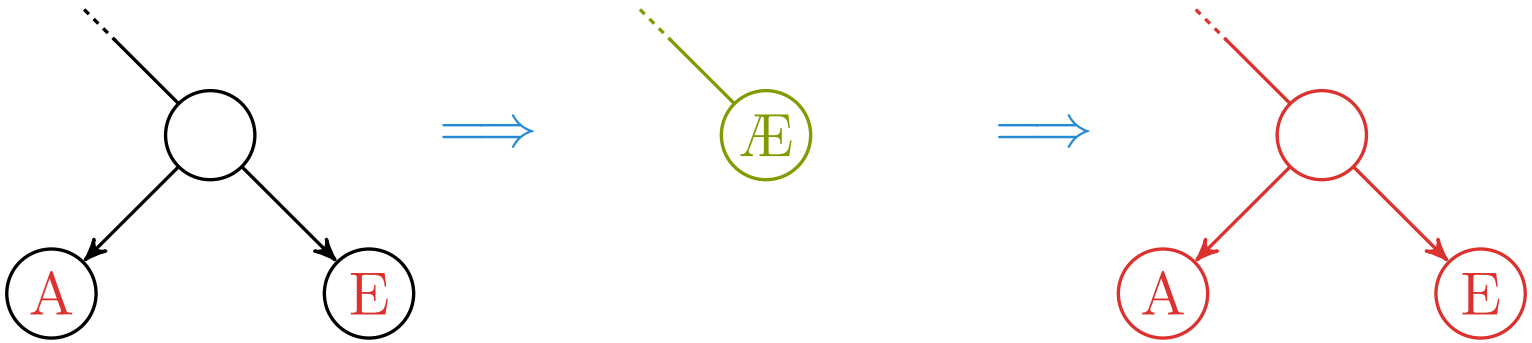
Løs det resulterende problemet



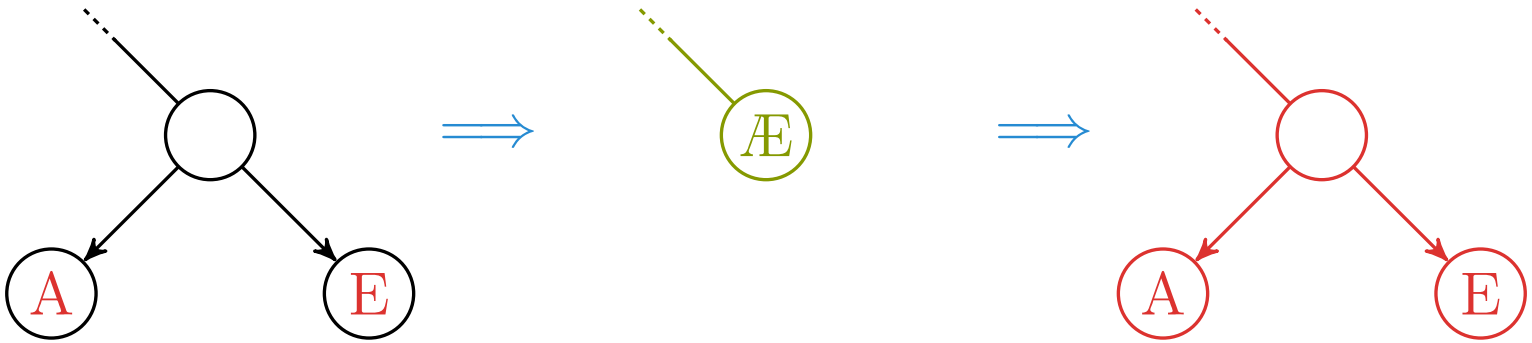
Del opp hybrid-tegnet igjen



Del opp hybrid-tegnet igjen



Suboptimal delløsning?



Suboptimal delløsning? Da kan vi forbedre løsningen!

- **Om vi velger grådig ...**
- **... og løser resten optimalt ...**
- **... så blir løsningen optimal.**
- **«Resten» har samme form som originalen**
- **Ved induksjon:**
  - **Vi kan velge grådig hele veien!**

Ofte kalt «Exchange arguments»

Bevis ved fortrinn

- Betrakt en vilkårlig løsning og transformér den gradvis til en grådig løsning, uten å senke kvaliteten. Den grådige løsningen må da være minst like god som enhver annen
- Prøv selv: Aktiviteter med varighet og deadline. Minimér største forsinkelse. Løsning: Velg hele tiden den med tidligst deadline
- Bevis optimalitet ved å starte med en optimal løsning, og så bytte om på elementer til du får den grådige. Hvordan unngår du å få en dårligere løsning?

Se http://www.idi.ntnu.no/~mlh/algkon/greedy.pdf

Optimum har ingen gaps. Hvis vi har en jobb som kommer senere men som har tidligere deadline, så vil vi også ha to jobber rett etter hverandre som er byttet om. Disse kan vi bytte, og få minst like bra resultat.

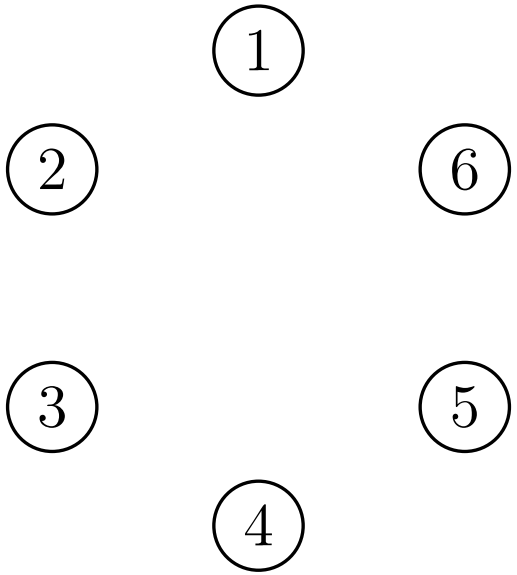
Velfundert induksjon

- **En relasjon er velfundert over en mengde hvis**
  alle ikke-tomme delmengder har minimale elementer
- **Ekvivalent: Ingen uendelige synkende kjeder**
- **Velfundert, endelig graf = rettet, asyklisk graf (DAG)**
- **Vi har alltid et sted å starte!**
- **Induksjon: Som sterk, naturlig induksjon med ny relasjon**
- **Lovlig ordning med avhengigheter:**
  Prosessér alltid dem med alle avhengigheter tilfredsstilt
  - **«Klipp av startnoder»**
  - **Topologisk sortering!**

Grafrepresentasjoner

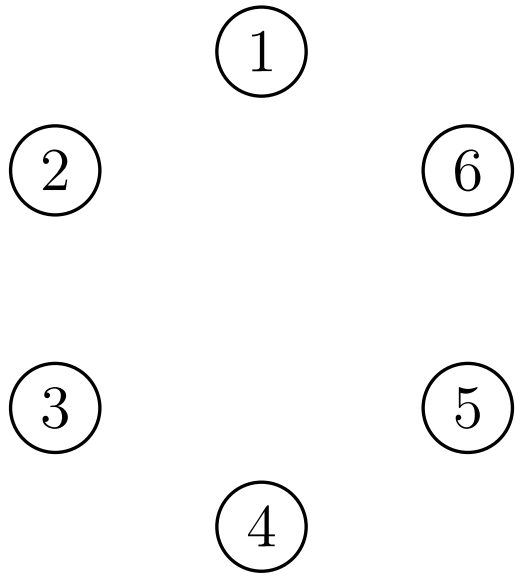
Nabomatriser

	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



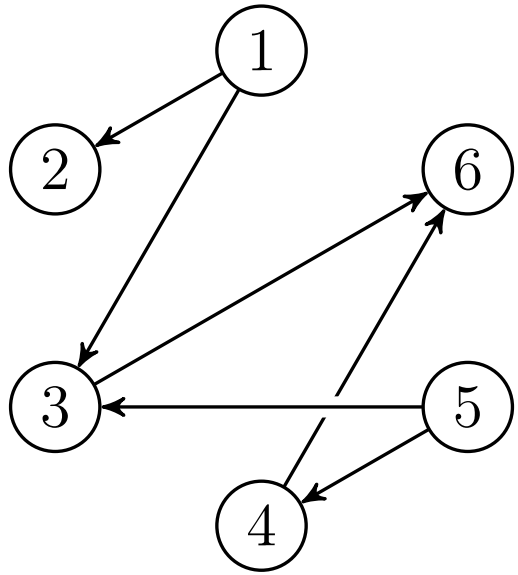
Nabomatrise: $A[u, v] \iff (u, v) \in G.E$

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



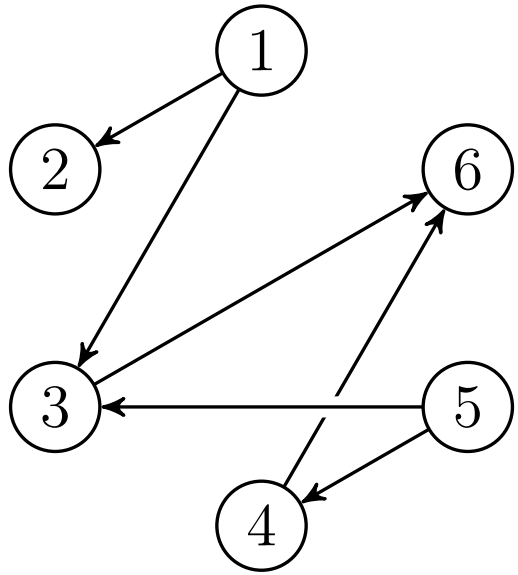
Nabomatrise: $A[u, v] \iff (u, v) \in G.E$

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



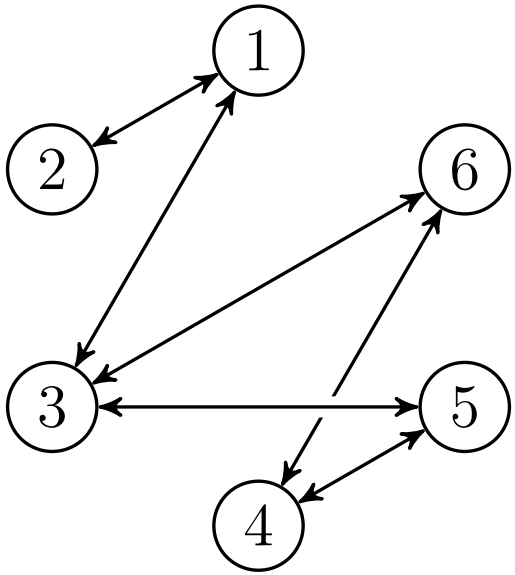
Nabomatrise: $A[u, v] \iff (u, v) \in G.E$

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



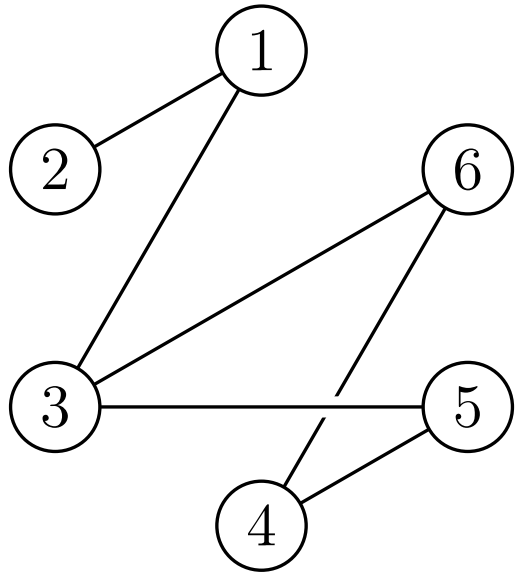
Urettet $G \iff$ symmetrisk A: $A[u, v] = A[v, u]$

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



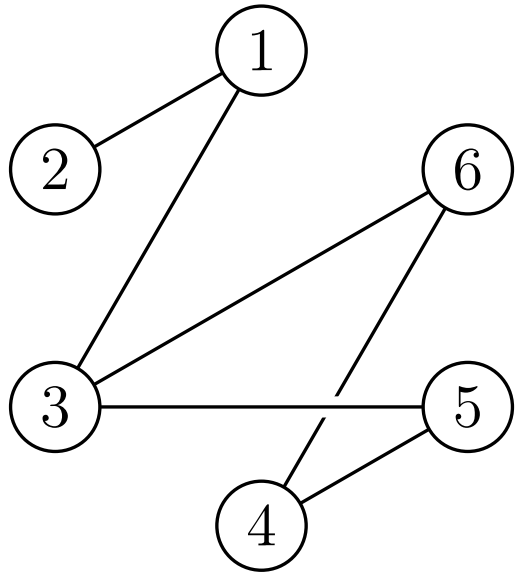
Urettet $G \iff$ symmetrisk A: $A[u, v] = A[v, u]$

	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: $A[u, v] = A[v, u]$

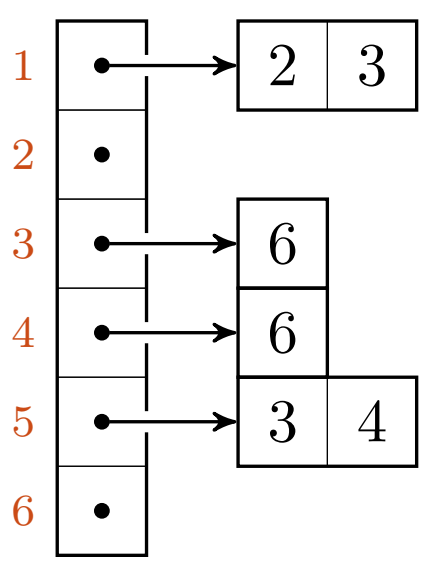
	1	2	3	4	5	6
1		1	1			
2	1					
3	1				1	1
4					1	1
5			1	1		
6			1	1		

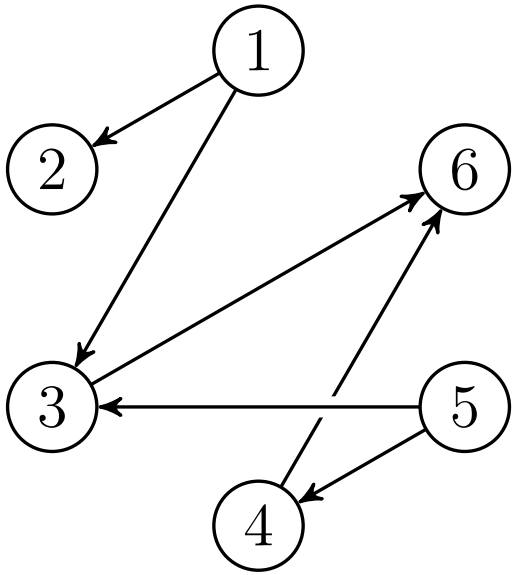


Egnet til raske oppslag; ikke så egnet til traversering

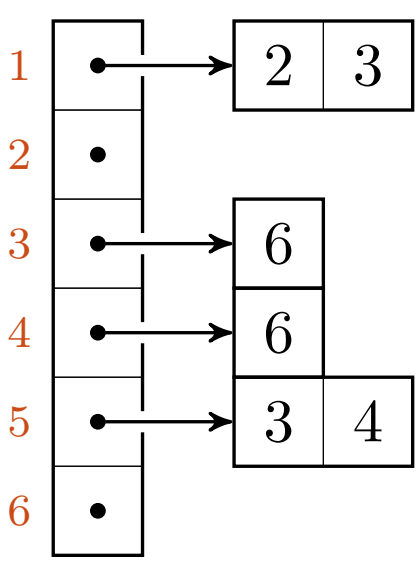
Nabolister

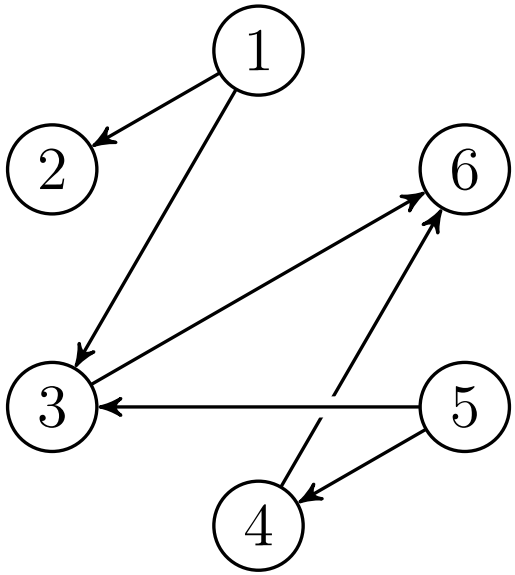
Nabolister: Liste (eller tabell) med ut-naboer for hver node



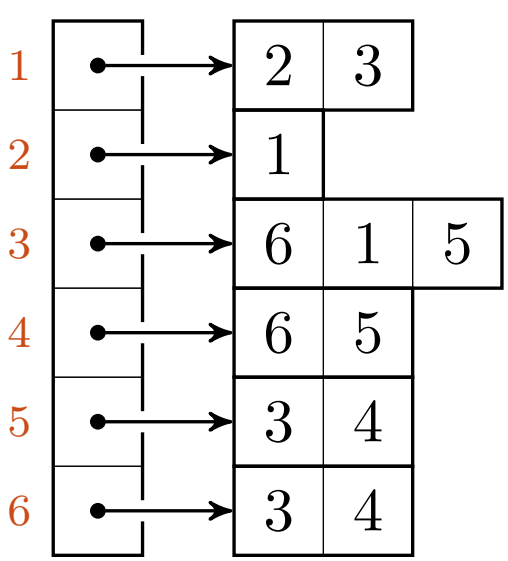


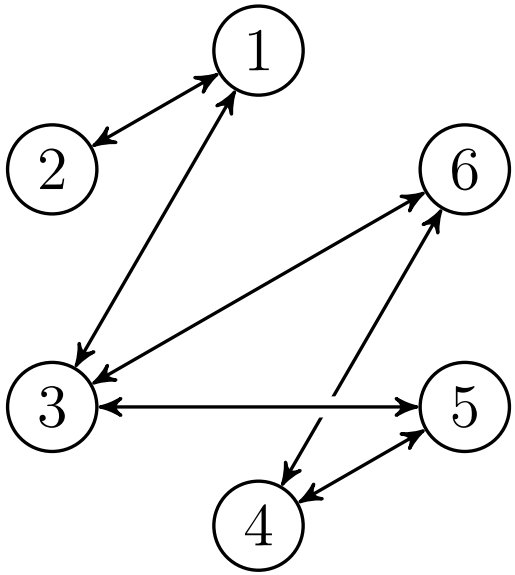
Nabolister: Liste (eller tabell) med ut-naboer for hver node



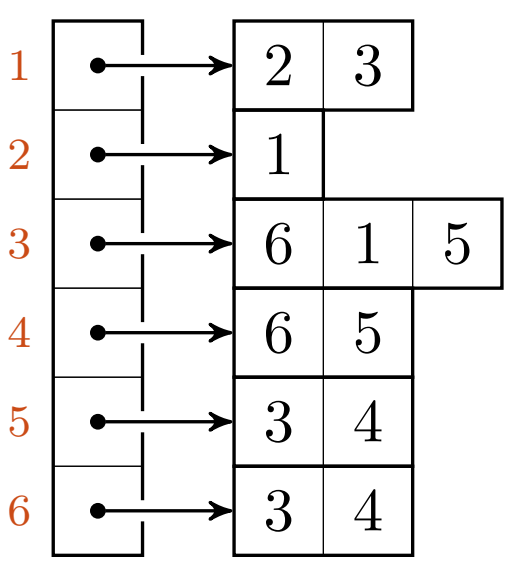


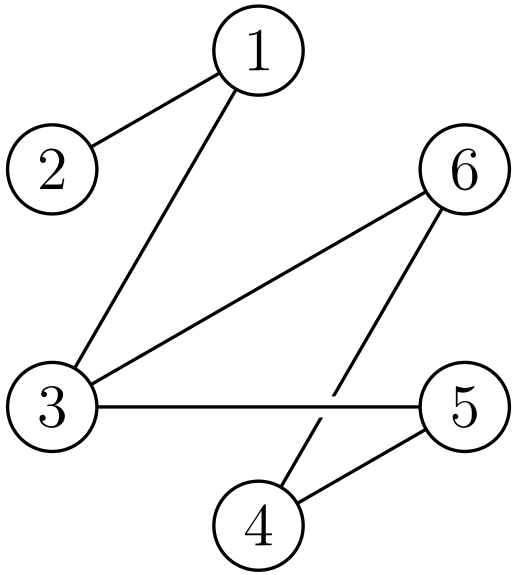
Urettet $G \iff$ kanter går begge veier



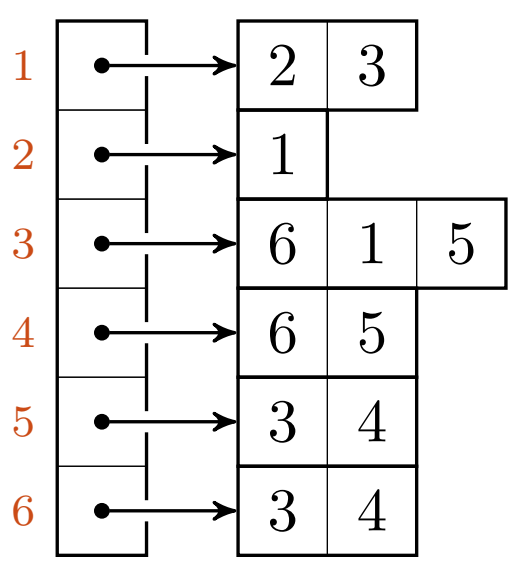


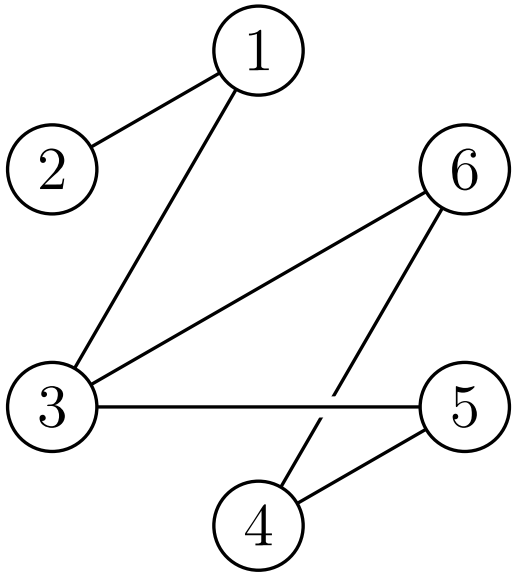
Urettet $G \iff$ kanter går begge veier





Urettet $G \iff$ kanter går begge veier





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

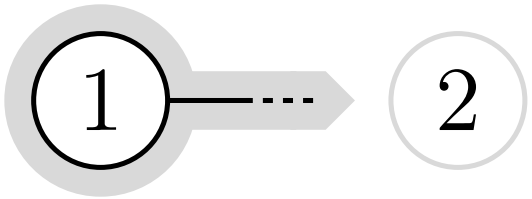
Kan godt «blande inn» hashtabeller e.l.

Traversering

Utforsking i lineær tid

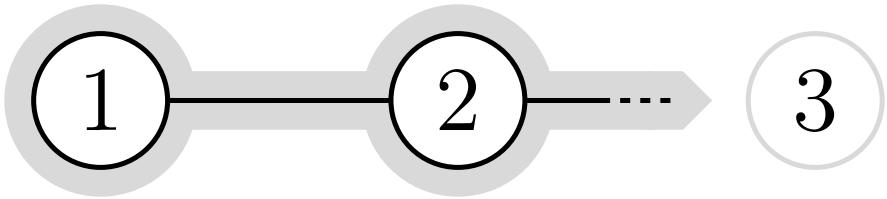
Traversering › **BFS**

Besøk node 1



Besøk node 1

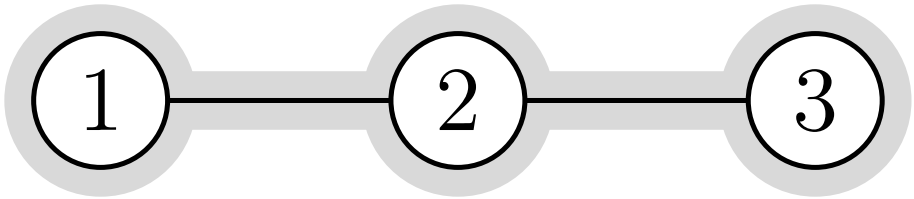
Besøk node 2



Besøk node 1

Besøk node 2

Besøk node 3

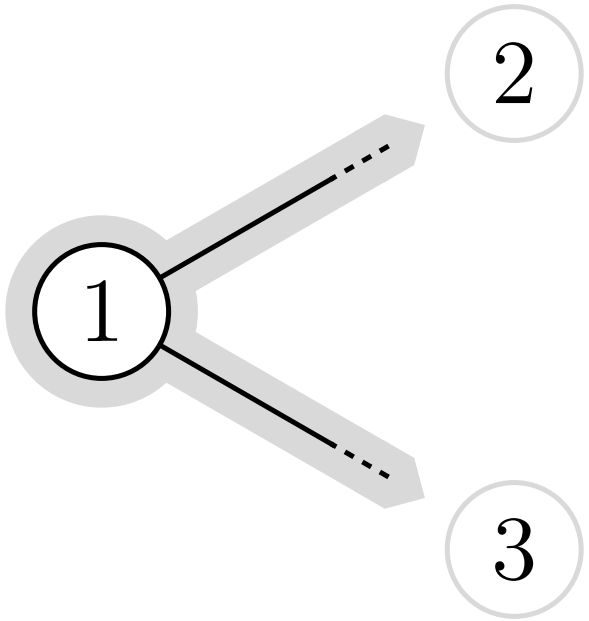


Besøk node 1

Besøk node 2

Besøk node 3

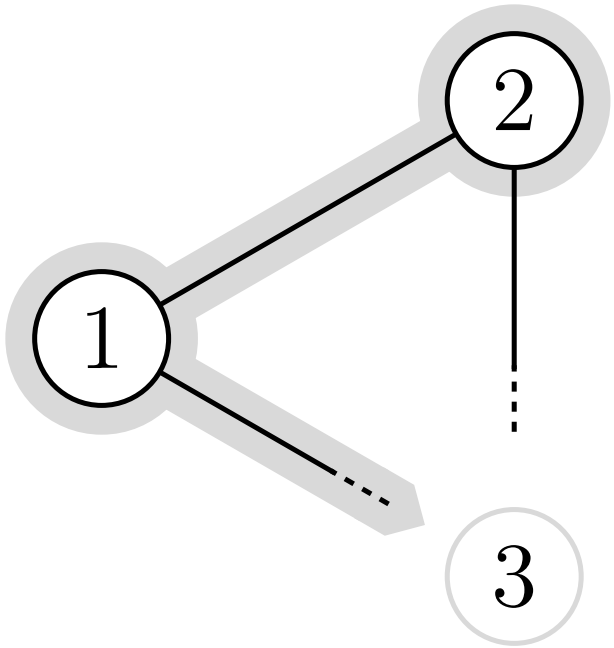
Besøk node 1



Besøk node 1

Besøk node 2

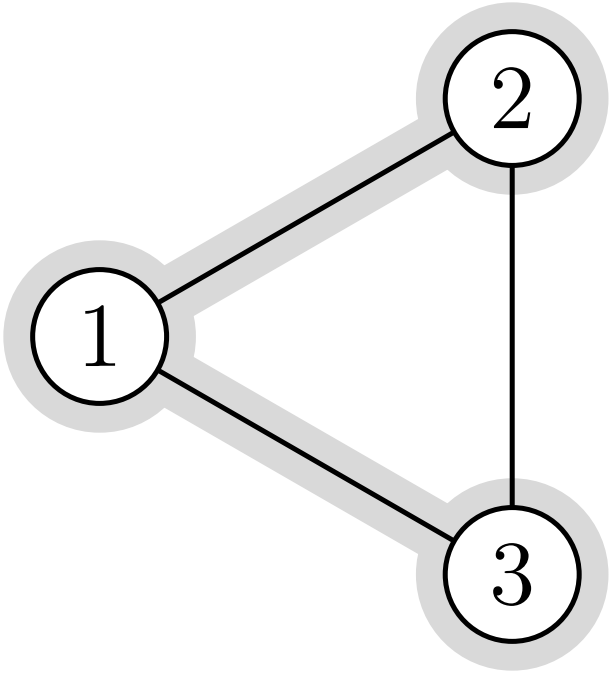
Besøk node 3



Besøk node 1

Besøk node 2

Besøk node 3

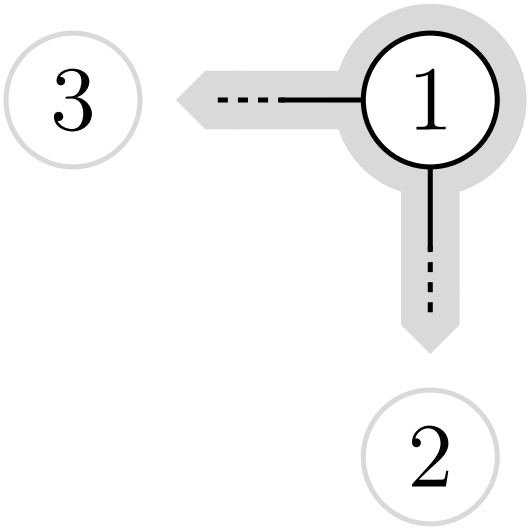


Besøk node 1

Besøk node 2

Besøk node 3

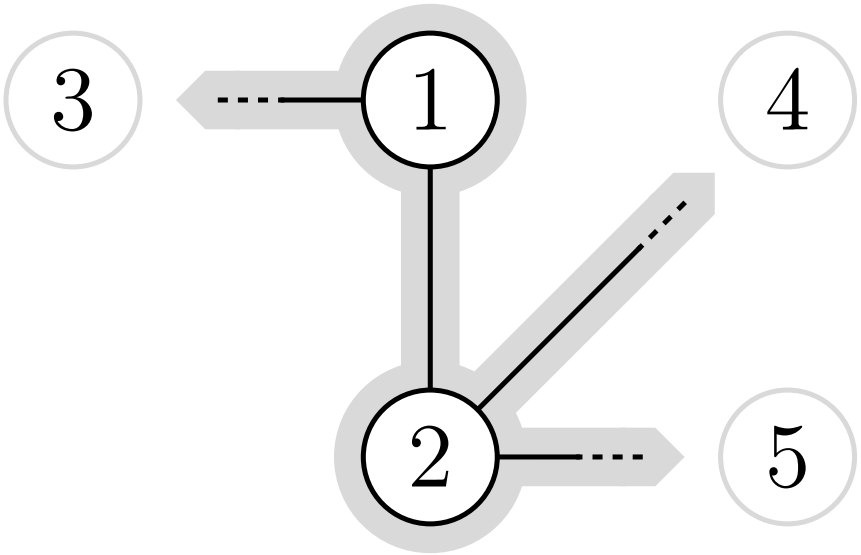
Besøk node 1



Besøk node 1

Besøk node 2

Besøk node 3



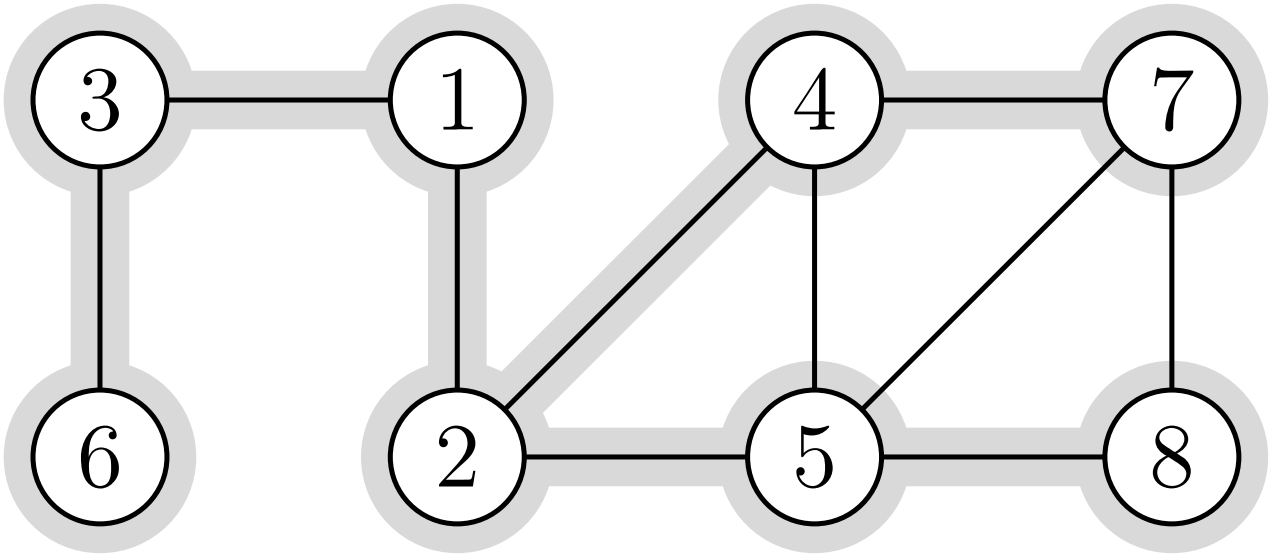
Besøk node 1

Besøk node 2

Besøk node 3

Besøk node 4

Besøk node 5



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

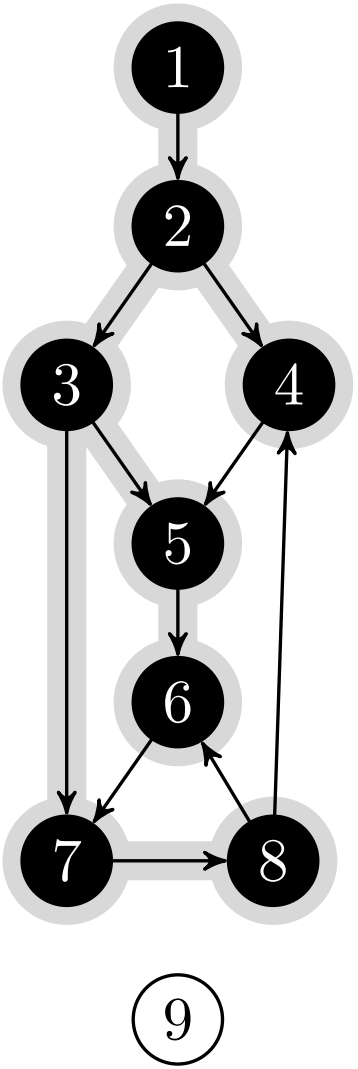
```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2        u.color = WHITE
 3        u.d = ∞
 4        u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  ...
```

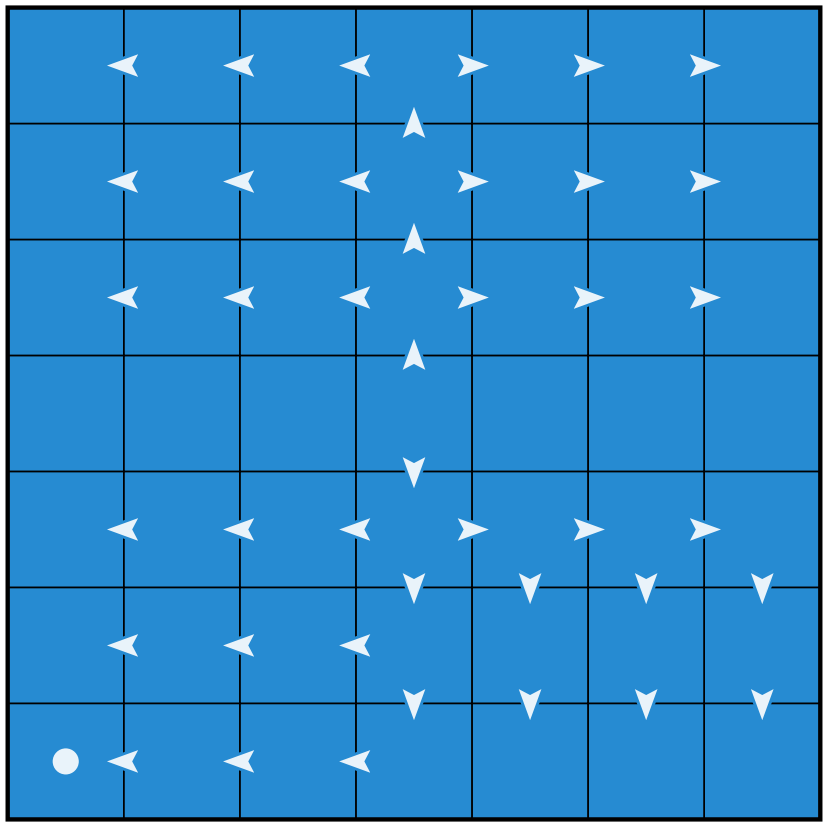
```
BFS(G, s)
 9  ...
10  while Q ≠ ∅
11        u = DEQUEUE(Q)
12        for each v ∈ G.Adj[u]
13              if v.color == WHITE
14                    v.color = GRAY
15                    v.d = u.d + 1
16                    v.π = u
17                    ENQUEUE(Q, v)
18        u.color = BLACK
```

```
BFS(G, s)
 9   ...
10   while Q ≠ ∅
11         u = DEQUEUE(Q)
12         for each v ∈ G.Adj[u]
13               if v.color == WHITE
14                     v.color = GRAY
15                     v.d = u.d + 1
16                     v.π = u
17                     ENQUEUE(Q, v)
18         u.color = BLACK
```

$u, v = 6, 7$

	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





BFS: Naboer stiller seg i kø

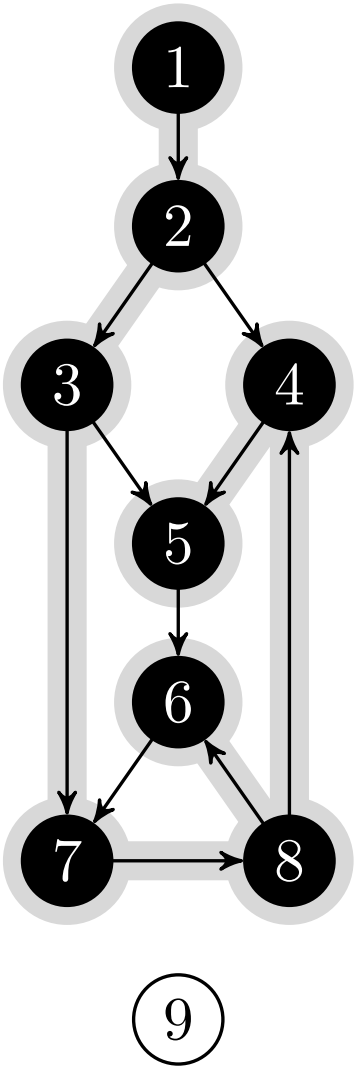
Iterativ DFS

Ikke essensielt å kunne detaljene i denne implementasjonen/ algoritmen – det viktige er å skjønne prinsippet med at DFS kan implementeres med en stakk.

```
ITER-DFS-VISIT(G, s)
 6   ...
 7   while Q ≠ ∅
 8         u = PEEK(Q)
 9         if u.color ≠ WHITE
10               u.color = BLACK
11               POP(Q)
12               continue
13         u.color = GRAY
14         for each v ∈ G.Adj[u]
15               if v.color == WHITE
16                     v.π = u
17                     PUSH(Q, v)
```

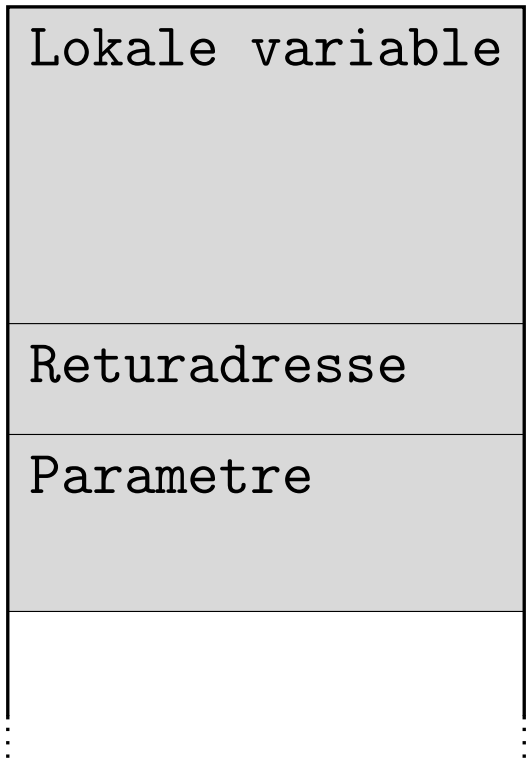
$u, v = 1, 6$

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



«Rekursjon ≈ stakk»

Implementeres vha. kallstakk



Tilstanden lagres midlertidig under funksjonskall

Traversering › DFS

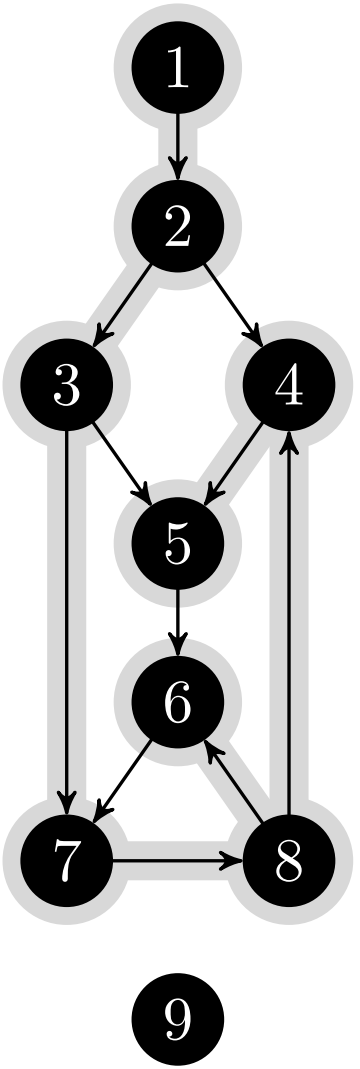
```
DFS(G)
1   for each vertex u ∈ G.V
2         u.color = WHITE
3         u.π = NIL
4   time = 0    # global
5   for each vertex u ∈ G.V
6         if u.color == WHITE
7               DFS-Visit(G, u)
```

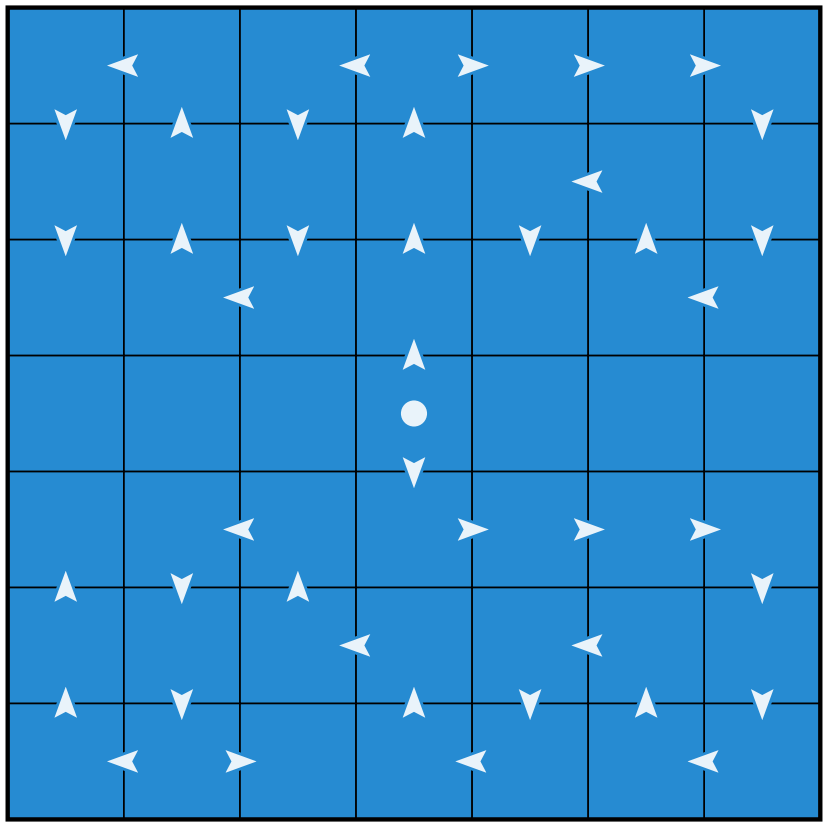
```
DFS-Visit(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ V.Adj[u]
 5        if v.color == WHITE
 6              v.π = u
 7              DFS-Visit(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

```
DFS(G)
1  for each vertex u ∈ G.V
2        u.color = WHITE
3        u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6        if u.color == WHITE
7              DFS-Visit(G, u)
```

$u, v = -, -$

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





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

Kantklassifisering

- **Tre-kanter**

  Kanter i dybde-først-skogen

- **Bakoverkanter**

  Kanter til en forgjenger i DF-skogen

- **Foroverkanter**

  Kanter utenfor DF-skogen to en etterkommer i DF-skogen

- **Krysskanter**

  Alle andre kanter

Kantklassifisering

- **Møter en hvit node**

  Tre-kant

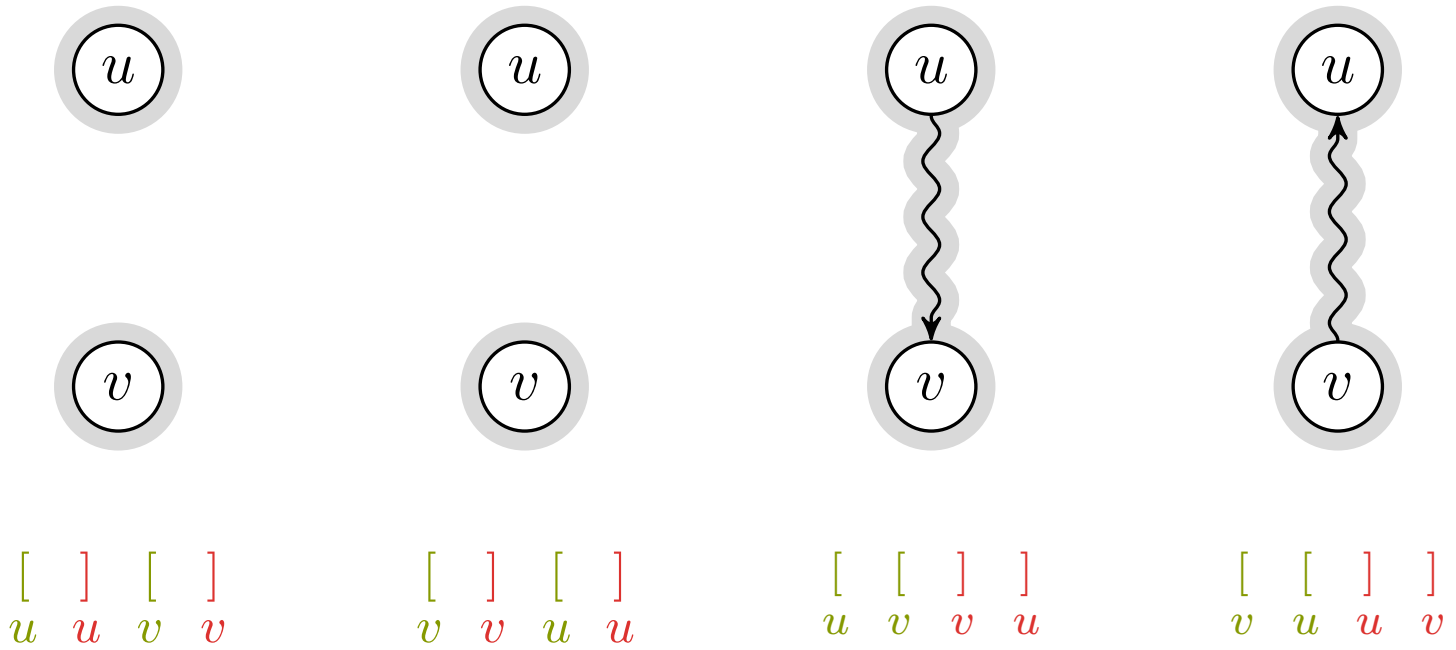
- **Møter en grå node**

  Bakoverkant

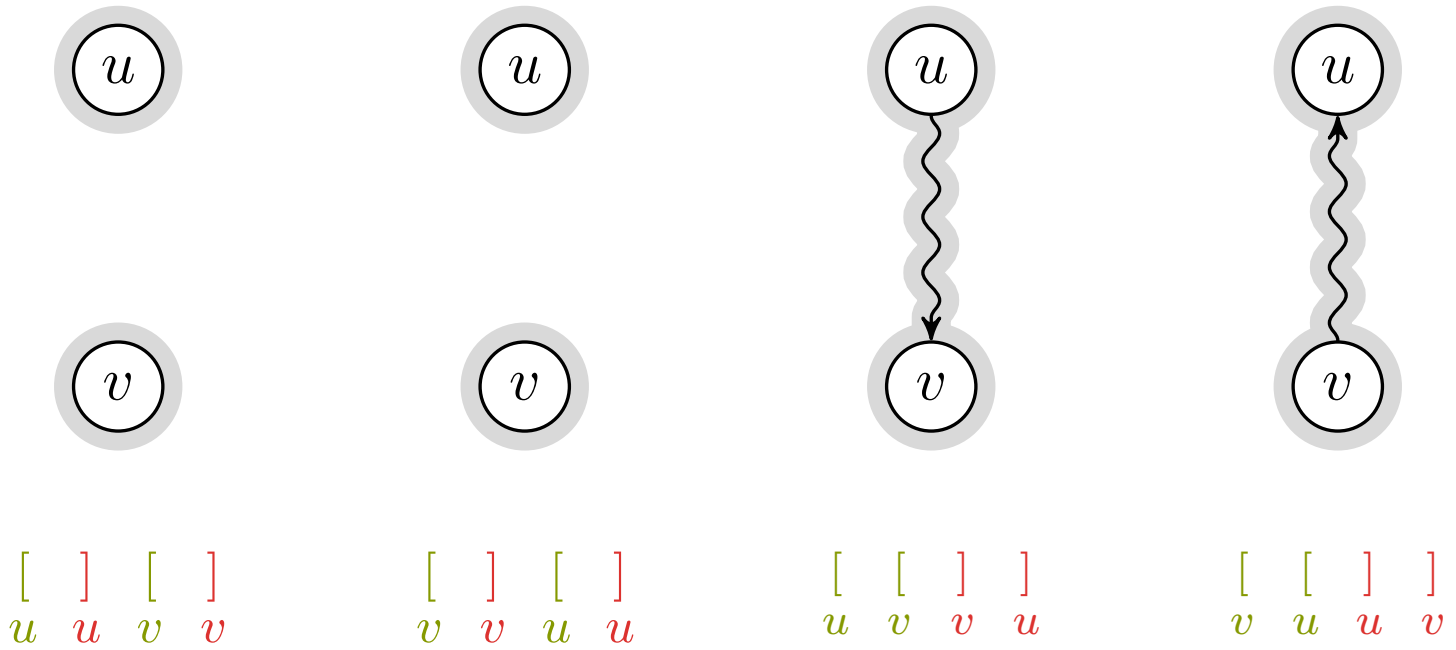
- **Møter en svart node:**

  Forover- eller krysskant

Parentesteoremet



Noder oppdages før og avsluttes etter sine etterkommere

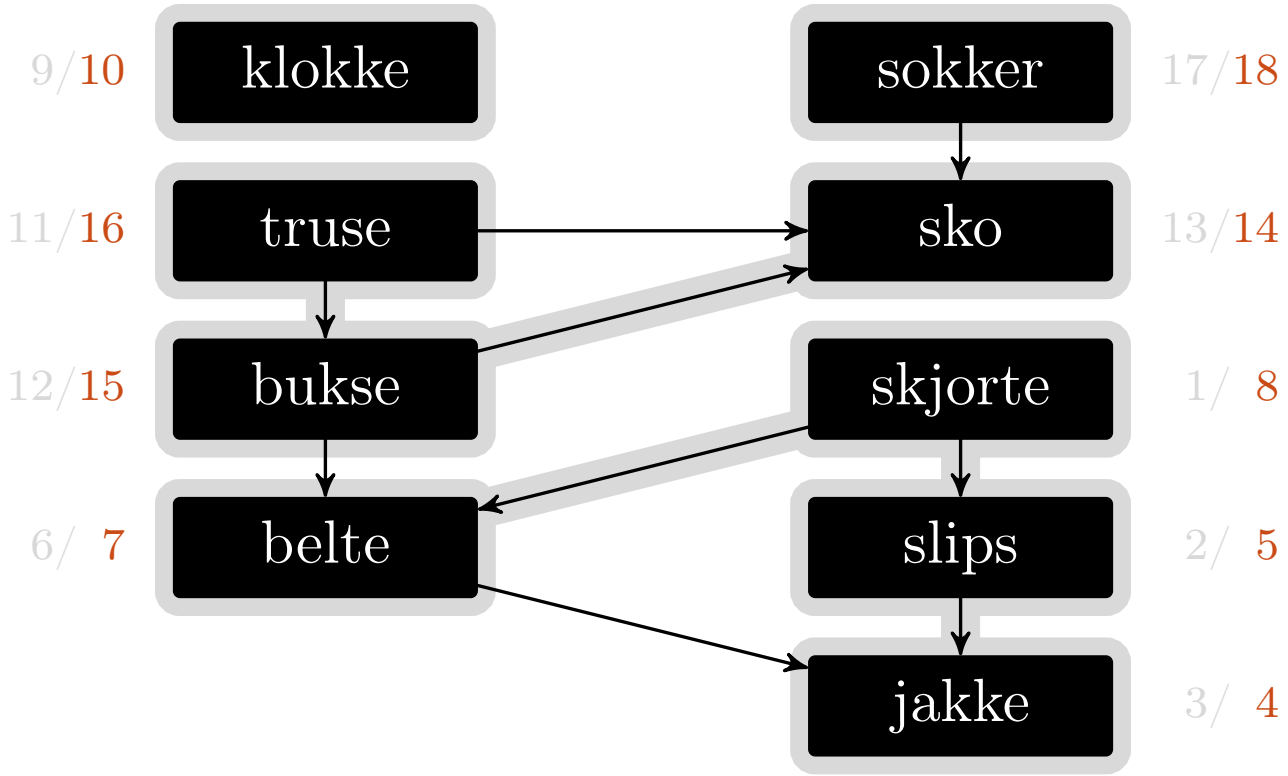


Dette er de eneste mulighetene!

Topologisk sortering

Topologisk sortering

- Gir nodene en rekkefølge
- Foreldre før barn
- Evt.: Alle kommer etter avhengigheter
- Krever DAG (dvs. velfundert)!


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

3/ 4

2/ 5

6/ 7

1/ 8

9/10

13/14

12/15

11/16

17/18