Forelesning 10

Bredde-først-søk kan finne stier med færrest mulig kanter, men hva om kantene har ulik lengde? Det generelle problemet er uløst, men vi kan løse problemet med gradvis bedre kjøretid for grafer uten negative sykler, uten negative kanter, og uten sykler. Og vi bruker samme prinsipp for alle tre!

Pensum

- [ ] Kap. 24. Single-source shortest paths: Innledning og 24.1–24.3

Læringsmål

[J_1] Forstå varianter av *korteste-vei*-problemet

[J_2] Forstå strukturen til problemet

[J_3] Forstå at negative sykler gir mening for korteste *enkle vei*

[J_4] Forstå at *korteste* og *lengste enkle vei* kan løses vha. hverandre

[J_5] Forstå *korteste-vei-trær*

[J_6] Forstå *kant-slakking* og Relax

[J_7] Forstå ulike egenskaper ved korteste veier og slakking

[J_8] Forstå Bellman-Ford

[J_9] Forstå Dag-Shortest-Path

[J_{10}] Forstå kobling mellom Dag-Shortest-Path og DP

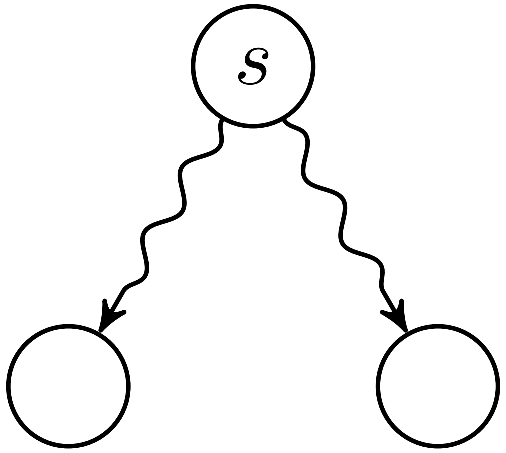
[J_{11}] Forstå Dijkstra

Forelesningen filmes

Forelesning 10

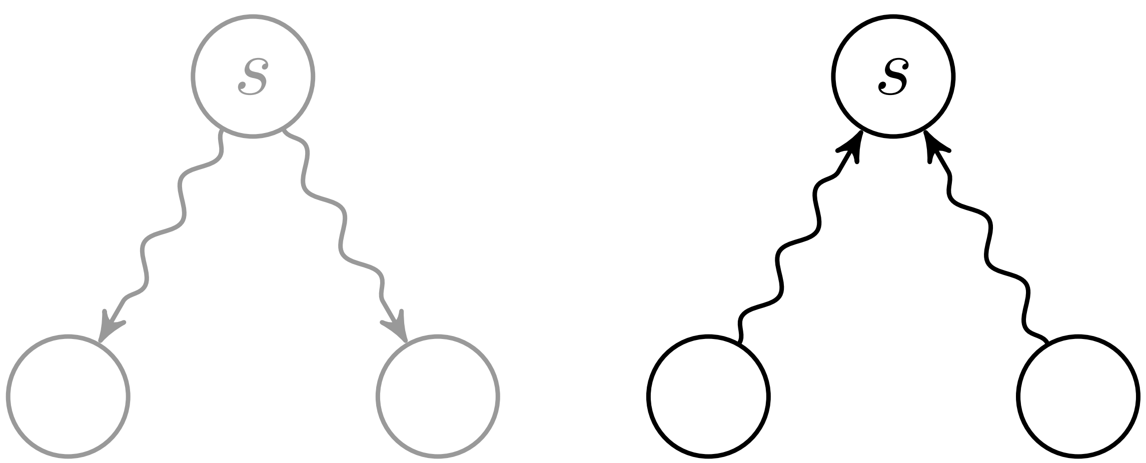
Korteste vei fra én til alle

1. **Dekomponering**
2. **DAG-Shortest-Path**
3. **Kantslakking**
4. **Bellman-Ford**
5. **Dijkstras algoritme**



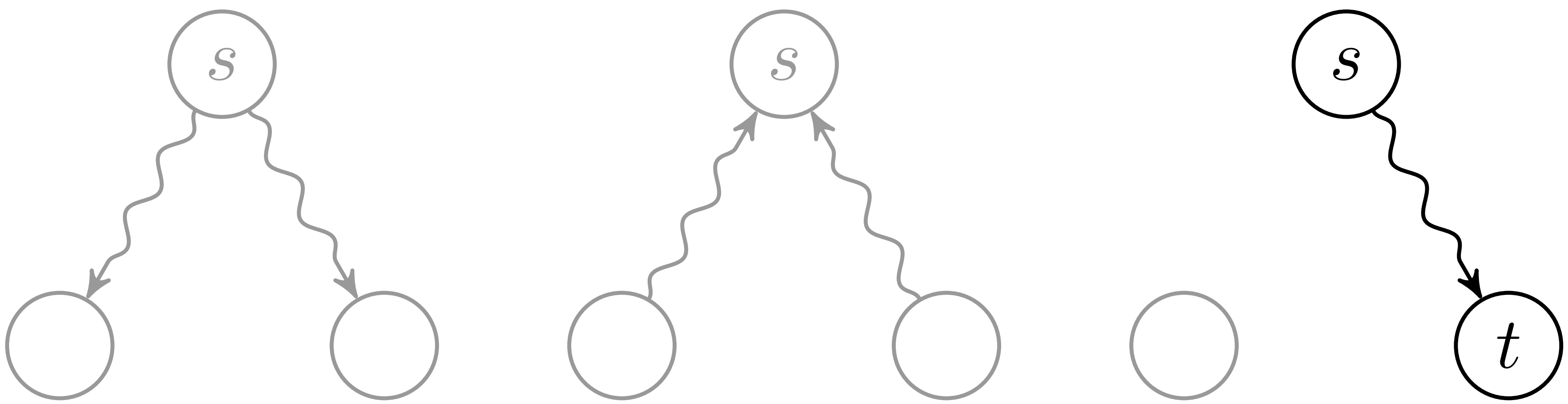
SSSP

Single source shortest path: Én til alle



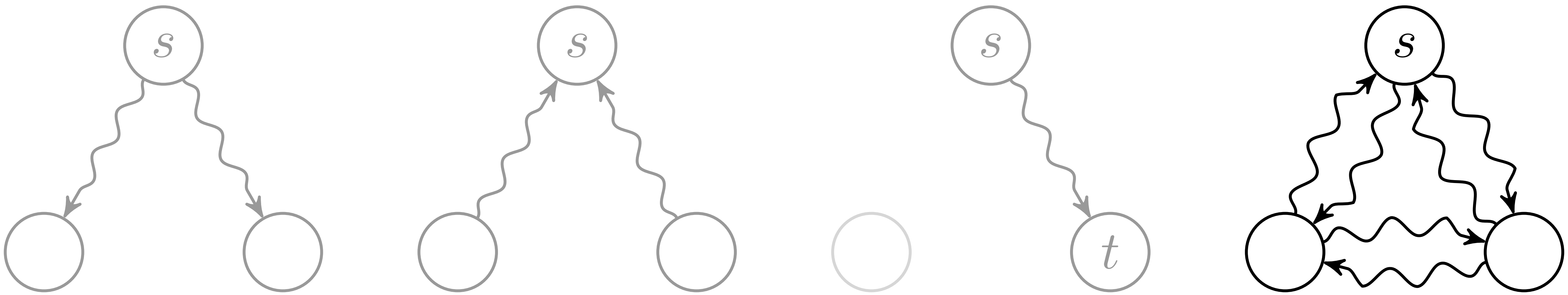
SSSP

Alle til én: Løses som SSSP i omvendt graf



SSSP

Én til én: Har ikke noe bedre enn SSSP



SSSP

APSP

All pairs shortest path: Alle til alle! (Neste gang)

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

Input: En rettet graf $G = (V, E)$, vektfunksjon $w : E \to \mathbb{R}$ og node $s \in V$.

Output: For hver node $v \in V$, en sti $p = \langle v_0, v_1, \ldots, v_k \rangle$ med $v_0 = s$ og $v_k = v$, som har minimal vektsum

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)\,.$$

Input: En rettet graf $G = (V, E)$, vektfunksjon $w : E \to \mathbb{R}$ og node $s \in V$.

Output: For hver node $v \in V$, en sti $p = \langle v_0, v_1, \ldots, v_k \rangle$ med $v_0 = s$ og $v_k = v$, som har minimal vektsum

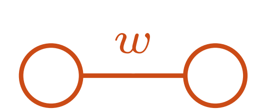
$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)\,.$$

Vi kaller også w lengde; $\delta(s, v) = w(p)$ er avstanden fra s til v

Input: En rettet graf $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, vektfunksjon $w : \mathrm{E} \rightarrow \mathbb{R}$ og node $s \in \mathrm{V}$.

Output: For hver node $v \in \mathrm{V}$, en sti $p = \langle v_0, v_1, \ldots, v_k \rangle$ med $v_0 = s$ og $v_k = v$, som har minimal vektsum

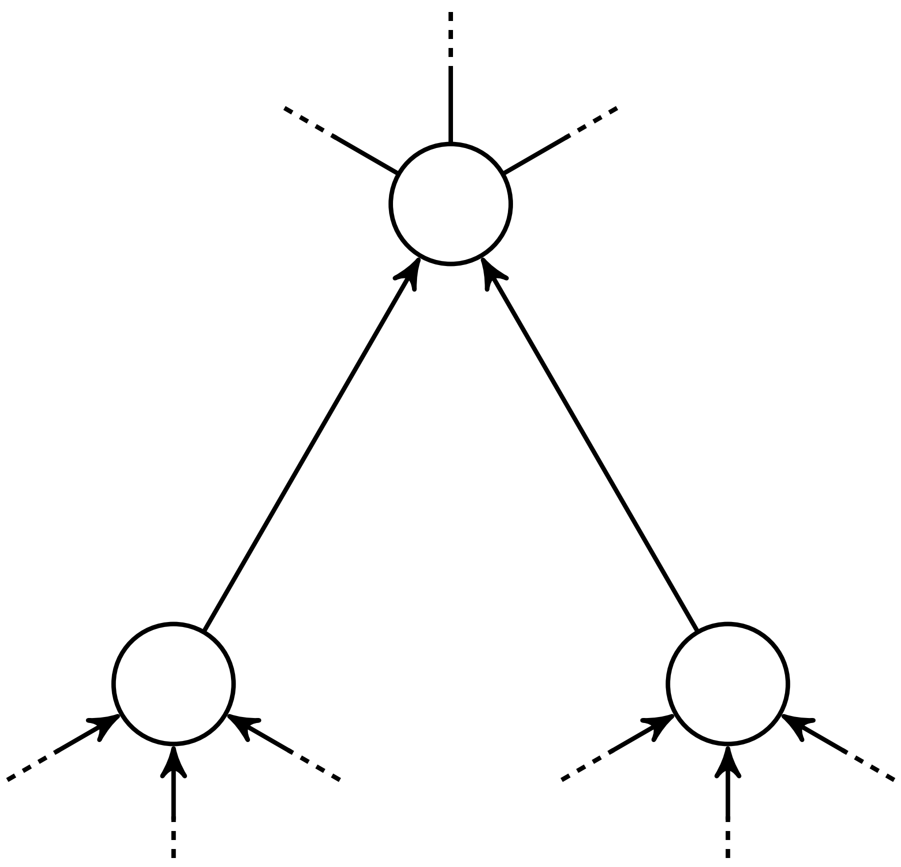
$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)\,.$$

Urettede grafer: 

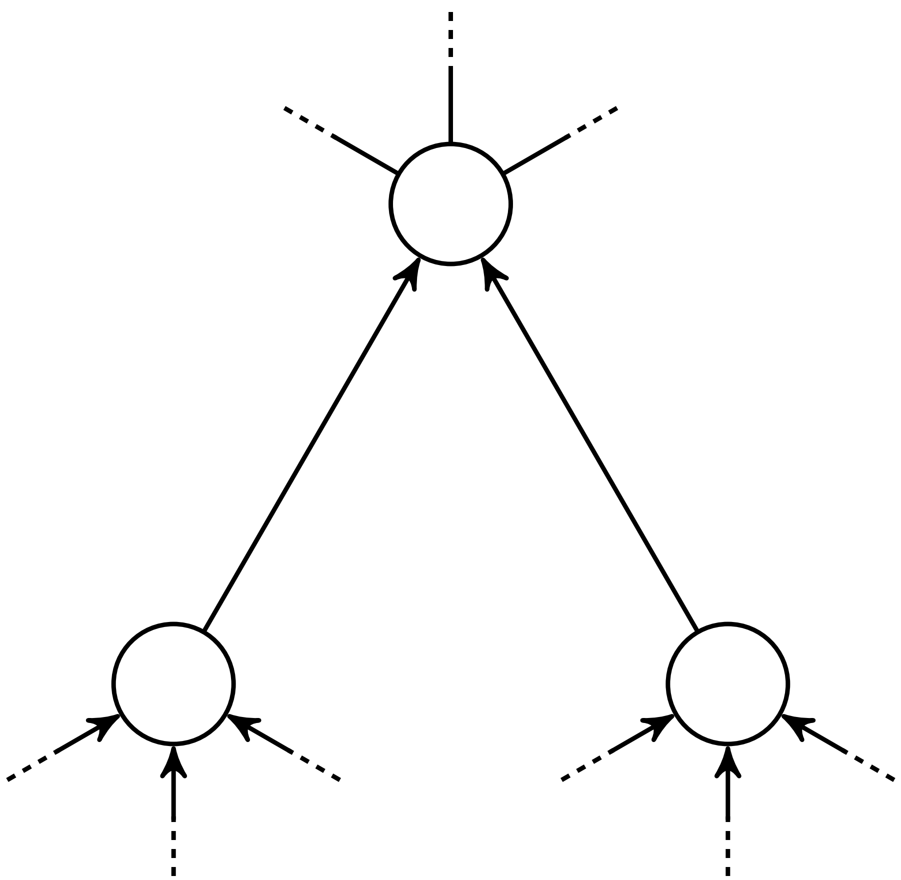
$\Longleftrightarrow$

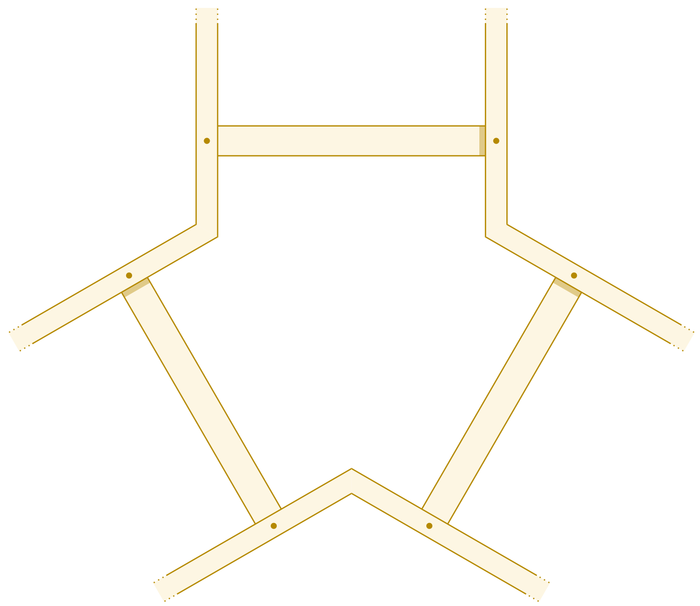
1:5

Dekomponering

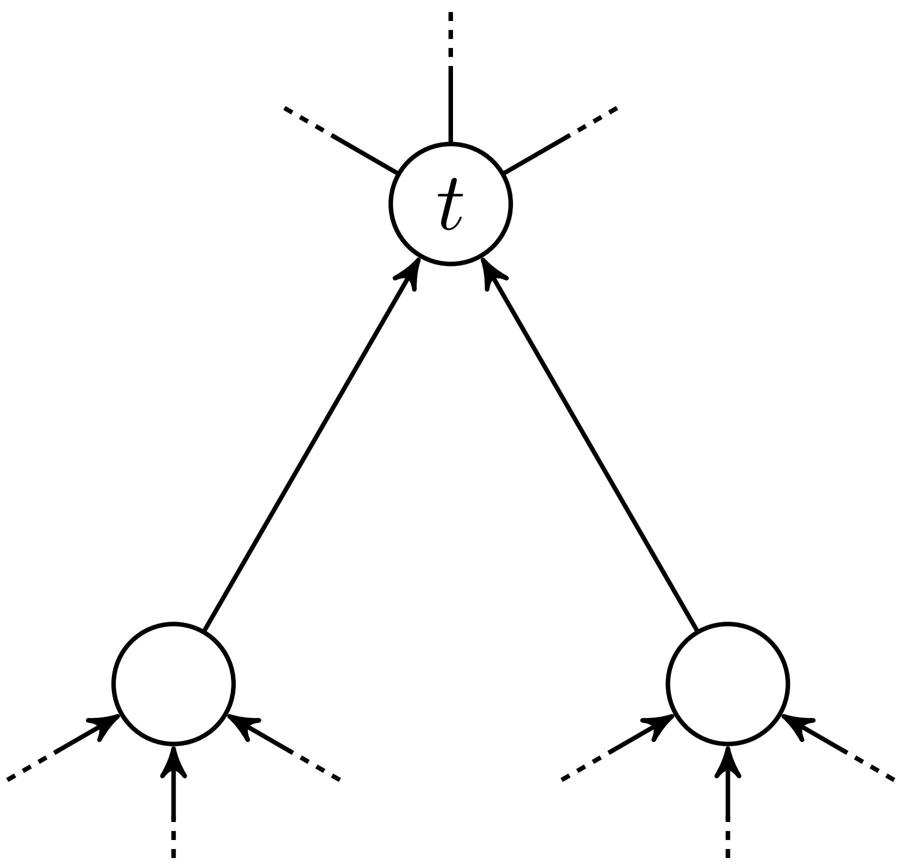


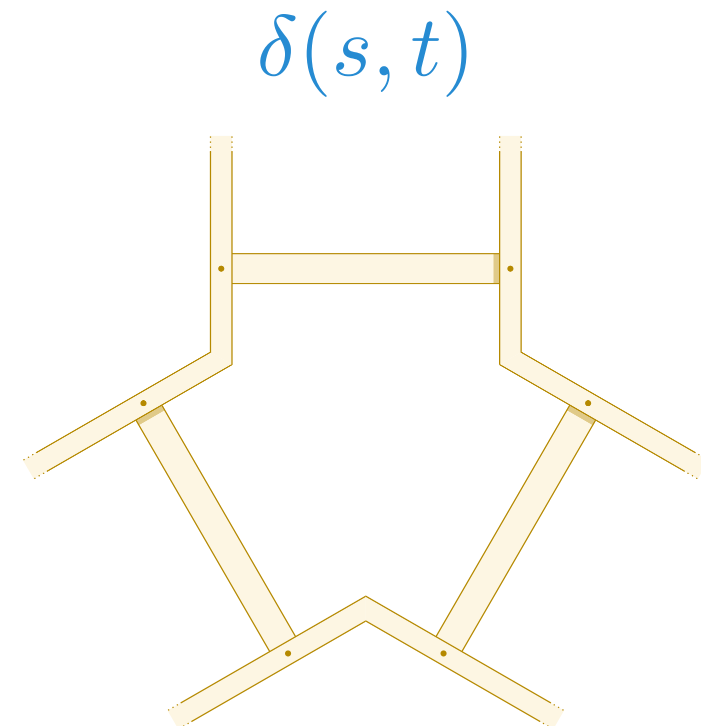
Vi begynner med å anta en asyklisk graf



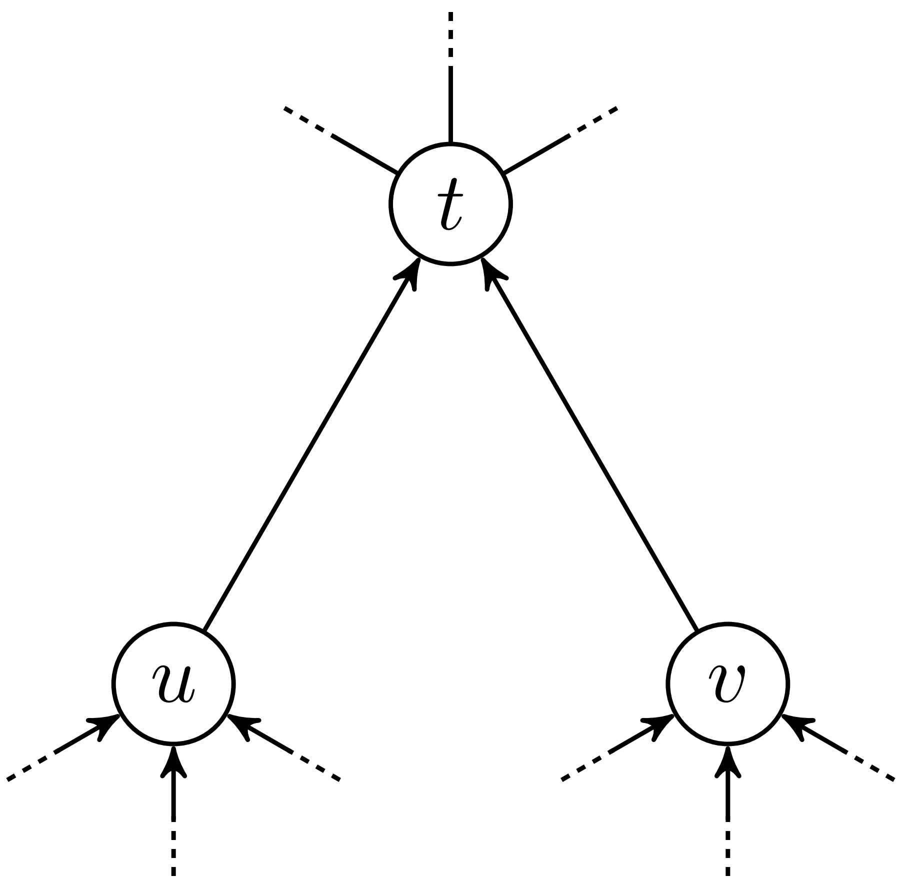


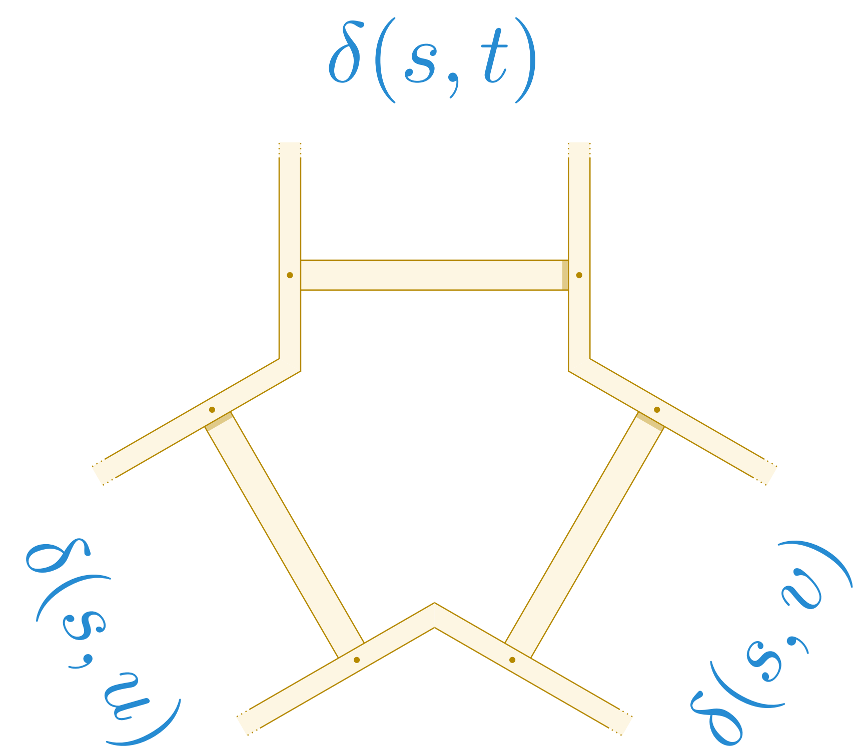
Vi kan da la grafen være sin egen delinstansgraf!



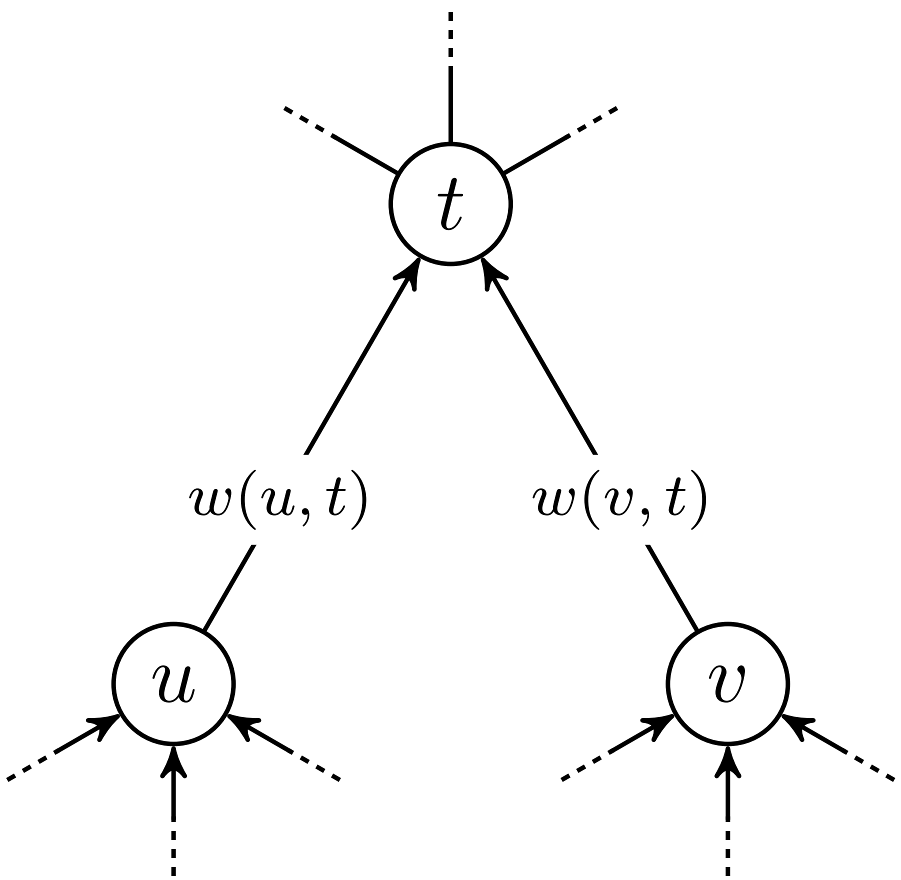


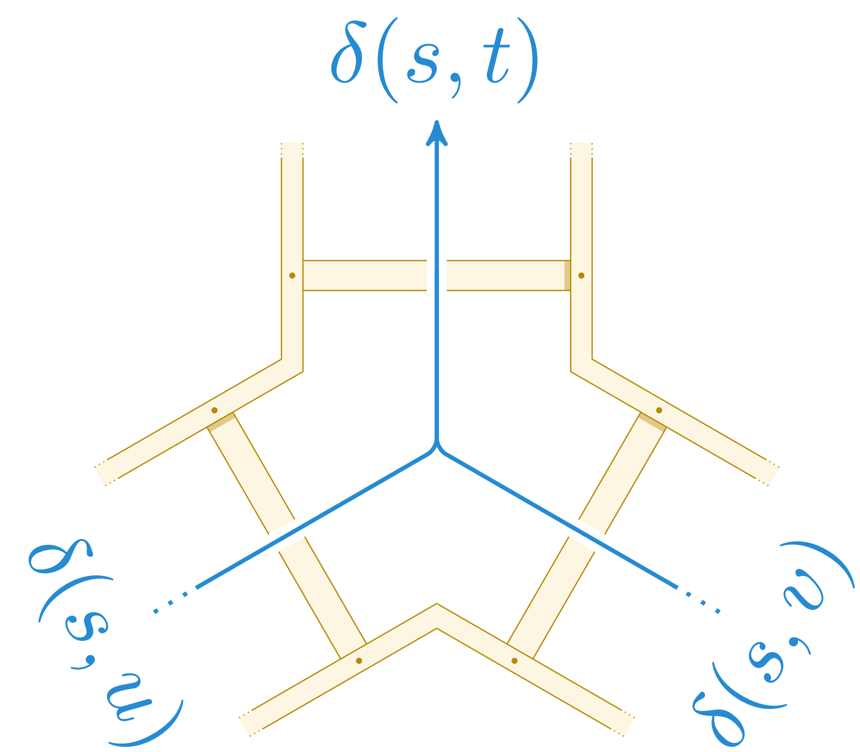
Vi vil finne avstand $\delta(s, t)$ fra startnoden s



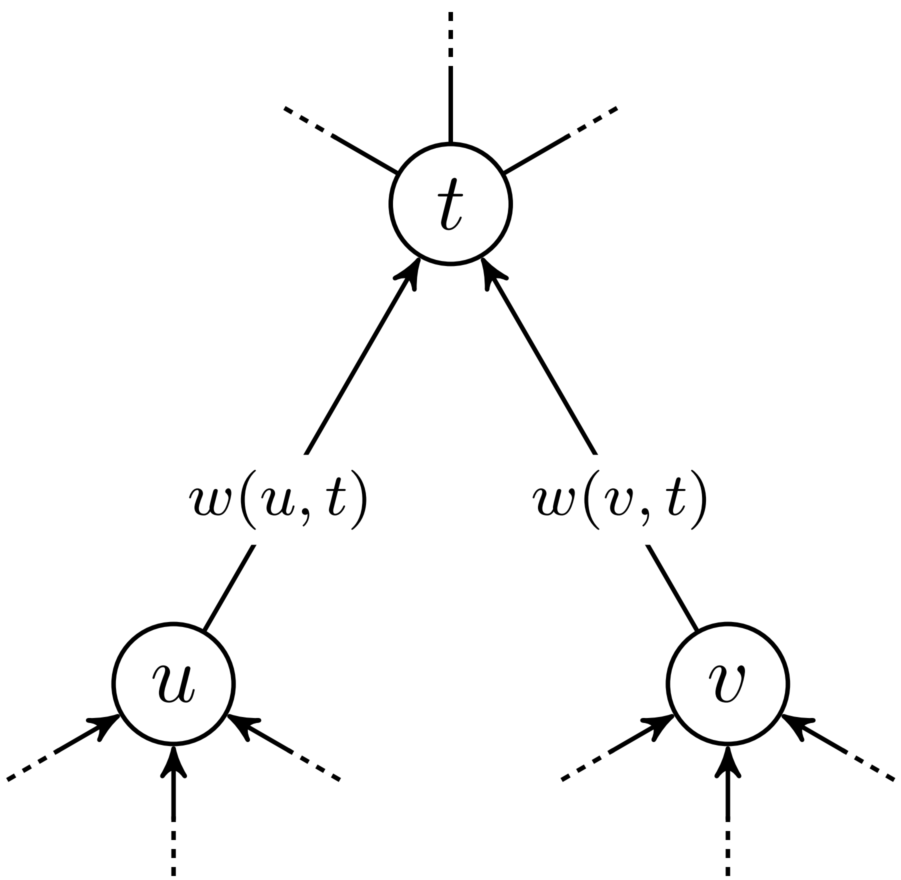


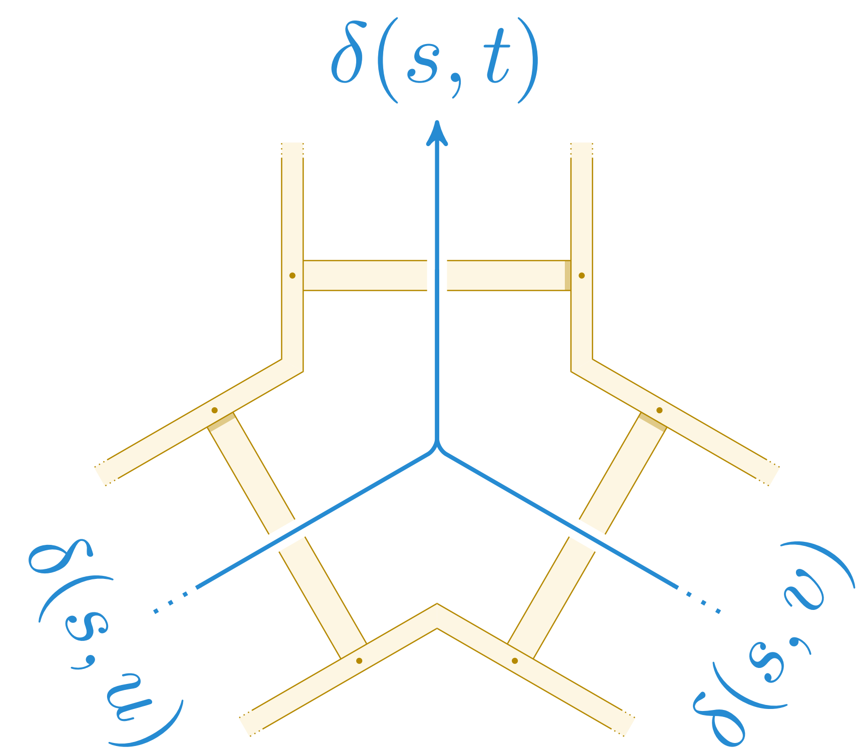
Vi antar induktivt at vi har funnet $\delta(s,-)$ for inn-naboer





Inn-nabo x gir mulig stilengde $\delta(s,x) + w(x,t)$; velg minimum!





Vi får her $\delta(s,t) = \min\{\, \delta(s,u) + w(u,t),\ \delta(s,v) + w(v,t) \,\}$

$min = \mathrm{A}[1]$
for $i = 2$ **to** $\mathrm{A}.length$
    if $min > \mathrm{A}[i]$
        $min = \mathrm{A}[i]$

Helt vanlig algoritme for å finne minimum (s. 213)

```
min = A[1]
for i = 2 to A.length
    if min > A[i]
        min = A[i]
```

```
min = ∞
for i = 1 to A.length
    if min > A[i]
        min = A[i]
```

Hvis vi har tilgang til ∞, trenger vi ikke ha A[1] i starten

$min = \mathrm{A}[1]$
for $i = 2$ **to** $\mathrm{A}.length$
    if $min > \mathrm{A}[i]$
        $min = \mathrm{A}[i]$

$min = \infty$
for each $x \in \mathrm{A}$
    if $min > x$
        $min = x$

Hvis vi har tilgang til ∞, trenger vi ikke ha A[1] i starten

$min = \mathrm{A}[1]$
for $i = 2$ **to** $\mathrm{A}.length$
  if $min > \mathrm{A}[i]$
    $min = \mathrm{A}[i]$

$min = \infty$
for each $x \in \mathrm{A}$
  if $min > x$
    $min = x$

Vi vil ha $t.d =$ minimum av $\delta(s, u) + w(u, t)$ for inn-naboer u

$t.d = \infty$
for each edge $(u, t) \in \mathrm{E}$
    if $t.d > \delta(s, u) + w(u, t)$
        $t.d = \delta(s, u) + w(u, t)$

Vi vil ha $t.d =$ minimum av $\delta(s, u) + w(u, t)$ for inn-naboer u

$t.d = \infty$
for each edge $(u, t) \in \mathrm{E}$
    if $t.d > \delta(s, u) + w(u, t)$
        $t.d = \delta(s, u) + w(u, t)$

Anta at vi allerede har funnet $u.d = \delta(s, u)$

$t.d = \infty$
for each edge $(u, t) \in \mathrm{E}$
    if $t.d > u.d + w(u, t)$
        $t.d = u.d + w(u, t)$

Anta at vi allerede har funnet $u.d = \delta(s, u)$

$t.d = \infty$
for each edge $(u, t) \in \mathrm{E}$
  if $t.d > u.d + w(u, t)$
    $t.d = u.d + w(u, t)$

For å løse problemet må vi gjøre dette for <u>alle</u> noder

mlh@ntnu.no

```
for each vertex v ∈ V
    v.d = ∞
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
```

For å løse problemet må vi gjøre dette for alle noder

for each vertex $v \in \mathrm{V}$
  $v.d = \infty$
  for each edge $(u, v) \in \mathrm{E}$
    if $v.d > u.d + w(u, v)$
      $v.d = u.d + w(u, v)$

Husk: $v.d$ er minimum av $u.d + w(u, v)$ for inn-naboer u

```
for each vertex v ∈ V
    v.d = ∞
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
```

Grunntilfelle: $s.d = 0$. Induktiv hypotese: $u.d$ er rett

```
for each vertex v ∈ V
    v.d = ∞
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
```

Induksjonen krever at $u.d$ beregnes før $v.d$

for each vertex $v \in \mathrm{V}$
    $v.d = \infty$
    for each edge $(u, v) \in \mathrm{E}$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Hvis $s.d = 0$ (grunntilfelle) og G er sortert topologisk . . .

```
topologically sort G
s.d = 0
for each vertex v ∈ V
    v.d = ∞
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
```

Hvis $s.d = 0$ (grunntilfelle) og G er sortert topologisk . . .

```
topologically sort G
s.d = 0
for each vertex v ∈ V
    v.d = ∞
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
```

. . . så følger $\delta(s, v) = v.d$ (for alle $v \in \mathrm{V}$), ved induksjon!

topologically sort G
$s.d = 0$
for each vertex $v \in V$
    $v.d = \infty$
    for each edge $(u, v) \in E$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

$$v.d = \min u.d + w(u, v)$$

Samme dekomponering fortsatt. Klassisk dynamisk programmering!

2:5

DAG-Shortest-Path

topologically sort G
$s.d = 0$
for each vertex $v \in V$
    $v.d = \infty$
    for each edge $(u, v) \in E$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Vi drar med oss info fra forgjengere, såkalt «pulling»

topologically sort G
$s.d = 0$
for each vertex $v \in \mathrm{V}$
    $v.d = \infty$
    for each edge $(u, v) \in \mathrm{E}$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Vi har ofte enklere tilgang til ut-naboer enn inn-naboer

mlh@ntnu.no

topologically sort G
$s.d = 0$
for each vertex $v \in V$
    $v.d = \infty$
    for each edge $(u, v) \in E$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Hvis vi skiller ut initialiseringen . . .

```
topologically sort G
for each vertex v ∈ V
        v.d = ∞
s.d = 0
for each vertex v ∈ V
        for each edge (u, v) ∈ E
                if v.d > u.d + w(u, v)
                        v.d = u.d + w(u, v)
```

Hvis vi skiller ut initialiseringen . . .

topologically sort G
for each vertex $v \in \mathrm{V}$
    $v.d = \infty$
$s.d = 0$
for each vertex $v \in \mathrm{V}$
    for each edge $(u, v) \in \mathrm{E}$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

…kan vi kjøre MINIMUM for flere noder, flettet sammen!

```
1  topologically sort G
2  for each vertex v ∈ V
3          v.d = ∞
4  s.d = 0
5  for each vertex v ∈ V
7          for each edge (u, v) ∈ E
8                  if v.d > u.d + w(u, v)
9                          v.d = u.d + w(u, v)
```

Så lenge $u.d$ er rett, kan vi trygt utføre linje 8 og 9

topologically sort G
for each vertex $v \in V$
    $v.d = \infty$
$s.d = 0$
for each vertex $v \in V$
    for each edge $(u, v) \in E$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Med andre ord, vi kan oppdatere (f.eks.) langs ut-kanter også!

topologically sort G
for each vertex $v \in \mathrm{V}$
    $v.d = \infty$
$s.d = 0$
for each vertex $u \in \mathrm{V}$
    for each edge $(u, v) \in \mathrm{E}$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Med andre ord, vi kan oppdatere (f.eks.) langs ut-kanter også!

topologically sort G
for each vertex $v \in \mathrm{V}$
    $v.d = \infty$
$s.d = 0$
for each vertex $u \in \mathrm{V}$
    for each edge $(u, v) \in \mathrm{E}$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Vi sprer da informasjon fra u til ut-naboene («reaching»)

topologically sort G
for each vertex $v \in \mathrm{V}$
    $v.d = \infty$
$s.d = 0$
for each vertex $u \in \mathrm{V}$
    for each edge $(u, v) \in \mathrm{E}$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$

Kravet fortsatt overholdt: (u, v) oppdateres når $u.d$ er korrekt

```
topologically sort G
for each vertex v ∈ V
    v.d = ∞
s.d = 0
for each vertex u ∈ V
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
```

Som vanlig i dynamisk programmering: Husk hvilket valg som tas

```
topologically sort G
for each vertex v ∈ V
    v.d = ∞
s.d = 0
for each vertex u ∈ V
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
```

I dette tilfellet: Hvilken forgjenger $v.\pi$ ga oss minimum, $v.d$?

topologically sort G
for each vertex $v \in \mathrm{V}$
    $v.d = \infty$
    $v.\pi = nil$
$s.d = 0$
for each vertex $u \in \mathrm{V}$
    for each edge $(u, v) \in \mathrm{E}$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$
            $v.\pi = u$

I dette tilfellet: Hvilken forgjenger $v.\pi$ ga oss minimum, $v.d$?

```
topologically sort G
for each vertex v ∈ V
        v.d = ∞
        v.π = nil
s.d = 0
for each vertex u ∈ V
        for each edge (u, v) ∈ E
                if v.d > u.d + w(u, v)
                        v.d = u.d + w(u, v)
                        v.π = u
```

Refaktorering: Trekk ut funksjoner!

```
topologically sort G
for each vertex v ∈ V
        v.d = ∞
        v.π = nil
s.d = 0
for each vertex u ∈ V
        for each edge (u, v) ∈ E
                if v.d > u.d + w(u, v)
                        v.d = u.d + w(u, v)
                        v.π = u
```

La oss kalle dette INITIALIZE-SINGLE-SOURCE(G, s)

```
topologically sort G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u ∈ V
    for each edge (u, v) ∈ E
        if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
            v.π = u
```

La oss kalle dette INITIALIZE-SINGLE-SOURCE(G, s)

topologically sort G
Initialize-Single-Source(G, s)
for each vertex $u \in V$
    for each edge $(u, v) \in E$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$
            $v.\pi = u$

La oss kalle dette Relax(u, v, w)

topologically sort G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex $u \in V$
    for each edge $(u, v) \in E$
        RELAX(u, v, w)

La oss kalle dette RELAX(u, v, w)

```
topologically sort G
Initialize-Single-Source(G, s)
for each vertex u ∈ V
    for each edge (u, v) ∈ E
        Relax(u, v, w)
```

Vi skal straks bruke «kantslakking» i en mer generell setting

mlh@ntnu.no

DAG-SHORTEST-PATH(G, w, s)

G	graf
w	vekting
s	startnode

Erke-eksempel på DP! Grafen er delproblemgrafen!

DAG-SHORTEST-PATH(G, w, s)
1 topologically sort the vertices of G

G graf
w vekting
s startnode

Alle noder v er avhengig av havner før v

DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)

G	graf
w	vekting
s	startnode

DAG-SHORTEST-PATH(G, w, s)
1 topologically sort the vertices of G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **for** each vertex u, in topsort order

G graf
w vekting
s startnode
u fra-node

«Bottom-up»-løsning

DAG-SHORTEST-PATH(G, w, s)
1 topologically sort the vertices of G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **for** each vertex u, in topsort order
4 **for** each vertex $v \in$ G.$Adj[u]$

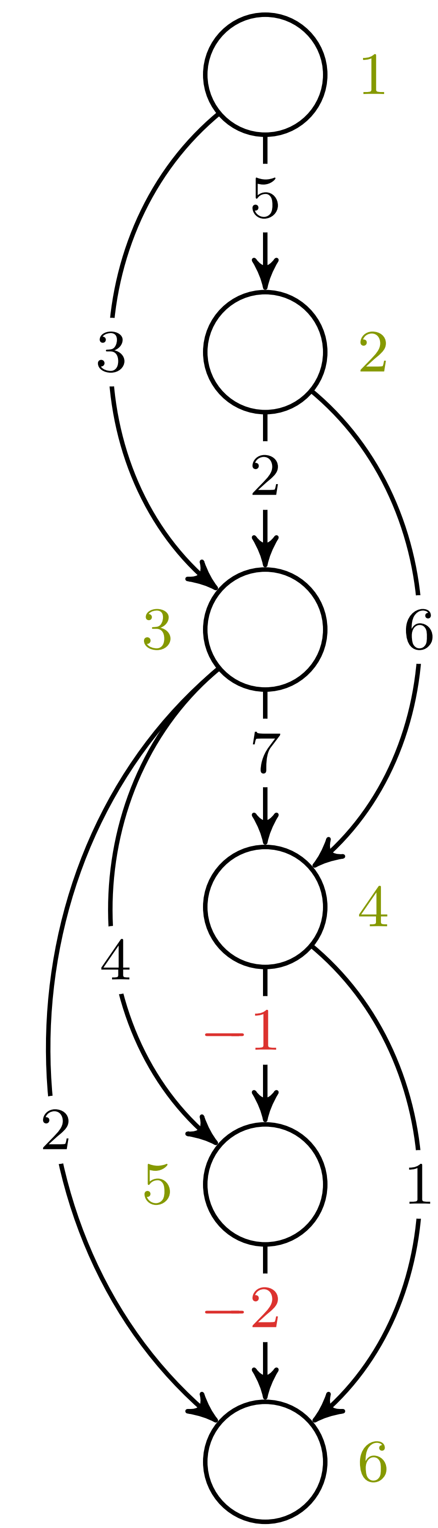
G graf
w vekting
s startnode
u fra-node
v til-node

```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

G	graf
w	vekting
s	startnode
u	fra-node
v	til-node

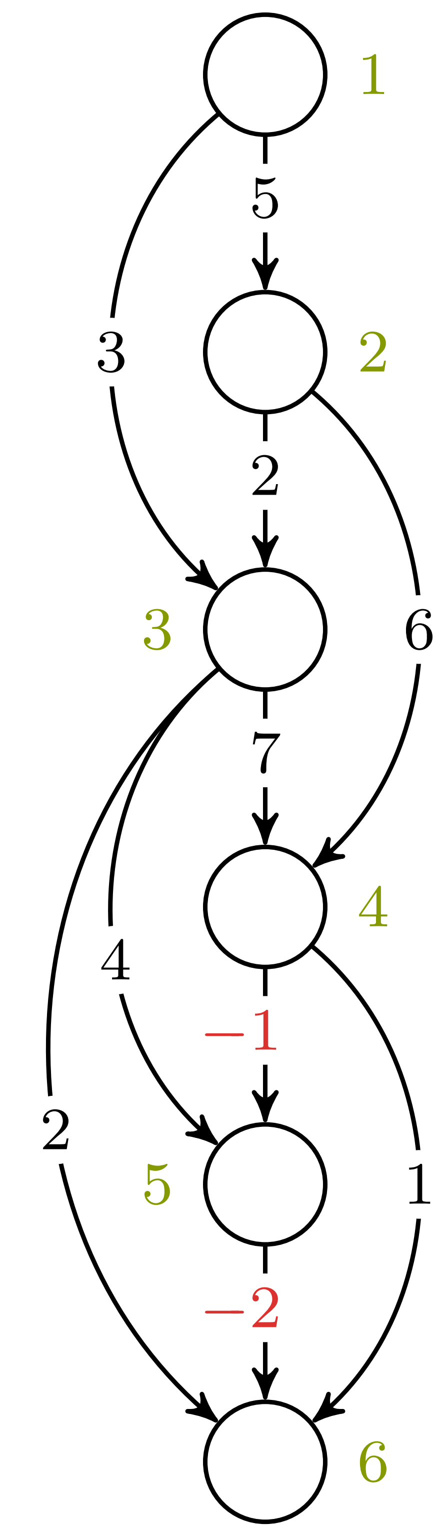
Må alt ha slakket inn-kanter; slakk alle ut-kanter

```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



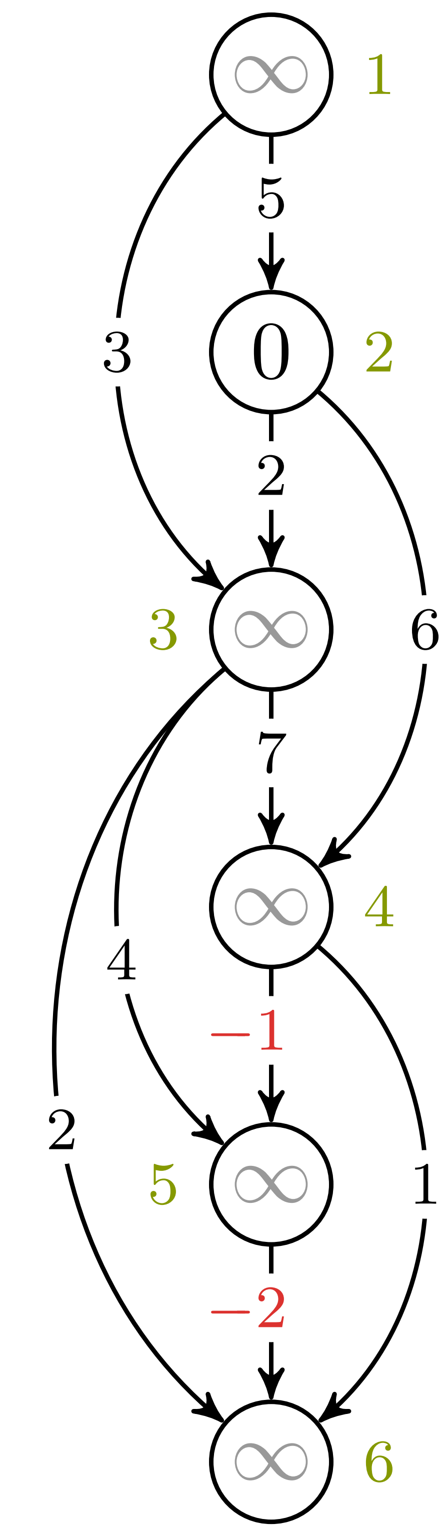
$s, u, v \;=\; 2, \text{-}, \text{-}$

DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w)



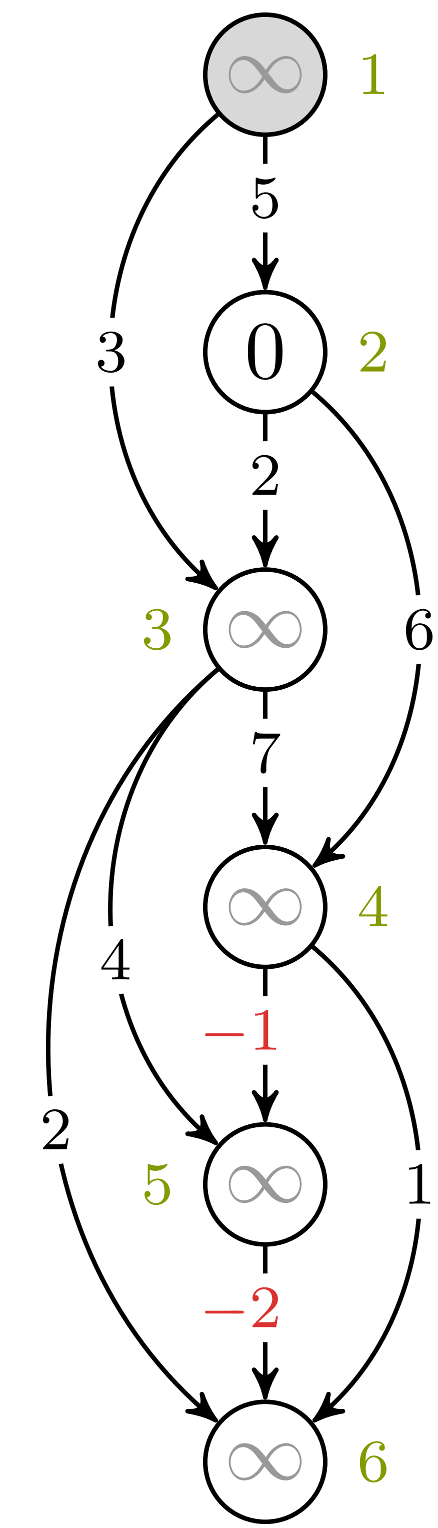
$s, u, v \; = \; 2, \text{-}, \text{-}$

```
DAG-SHORTEST-PATH(G, w, s)
  topologically sort the vertices of G
  INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex u, in topsort order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



$s, u, v \ = \ 2, \text{-}, \text{-}$

DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w)

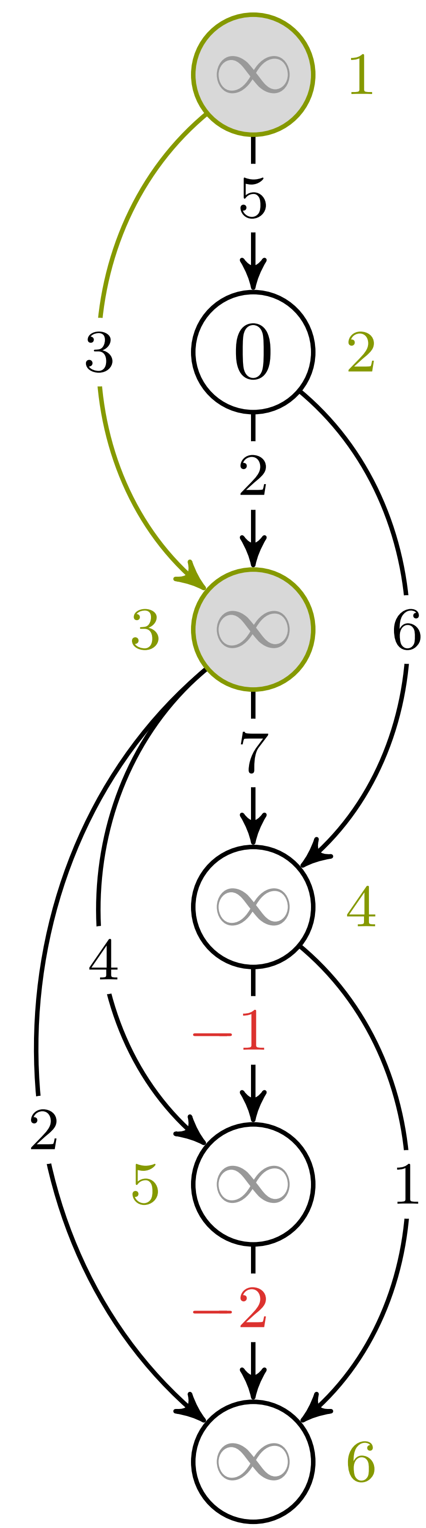


$s, u, v \;=\; 2, 1, \text{-}$

DAG-SHORTEST-PATH(G, w, s)

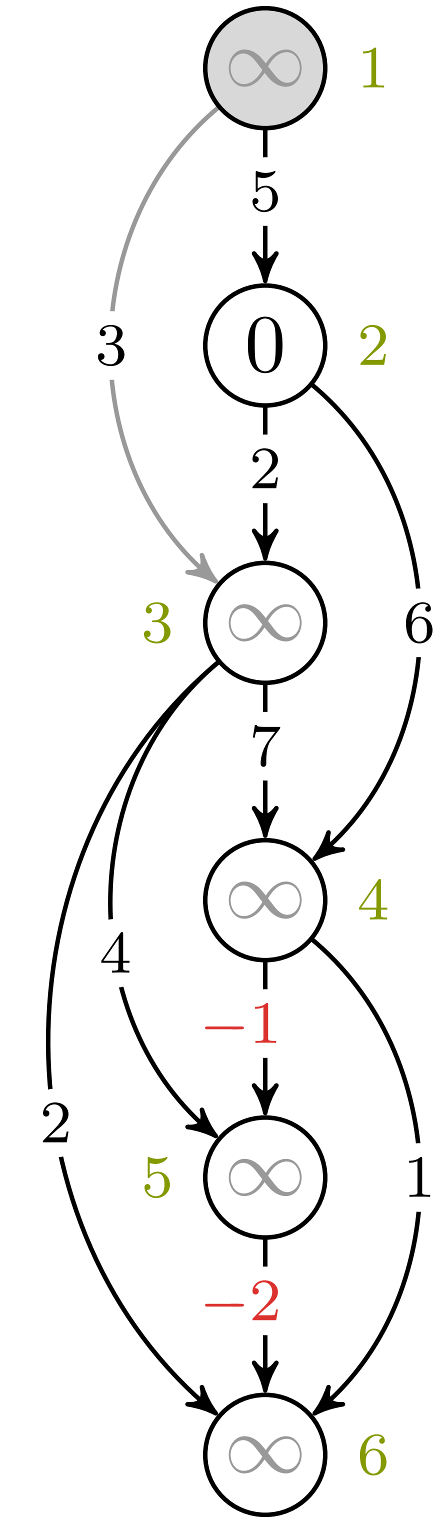
```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

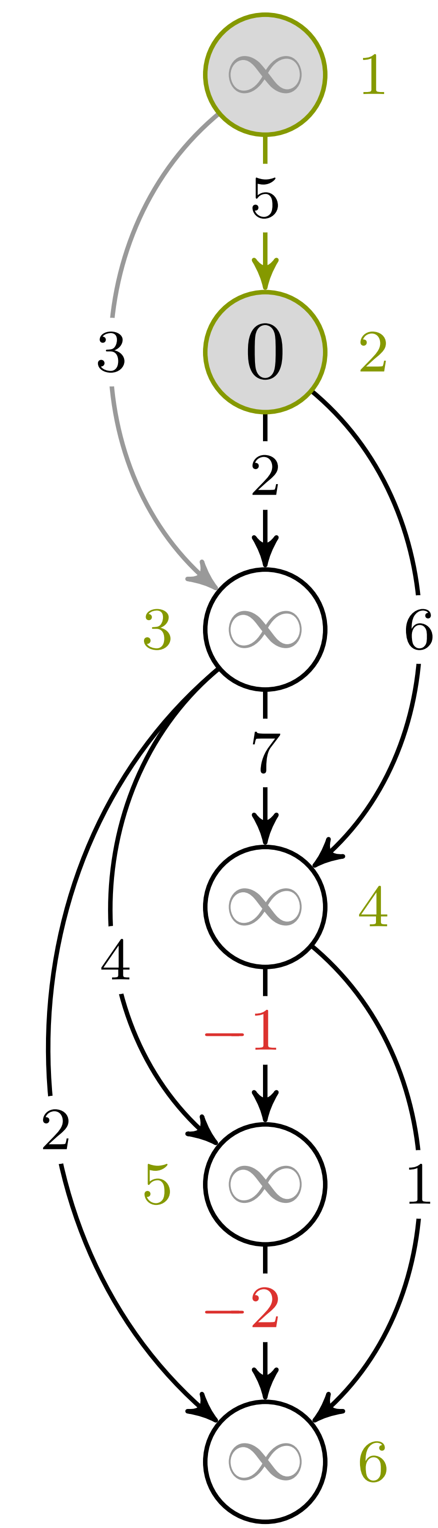
$s, u, v \;=\; 2, 1, 3$



```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

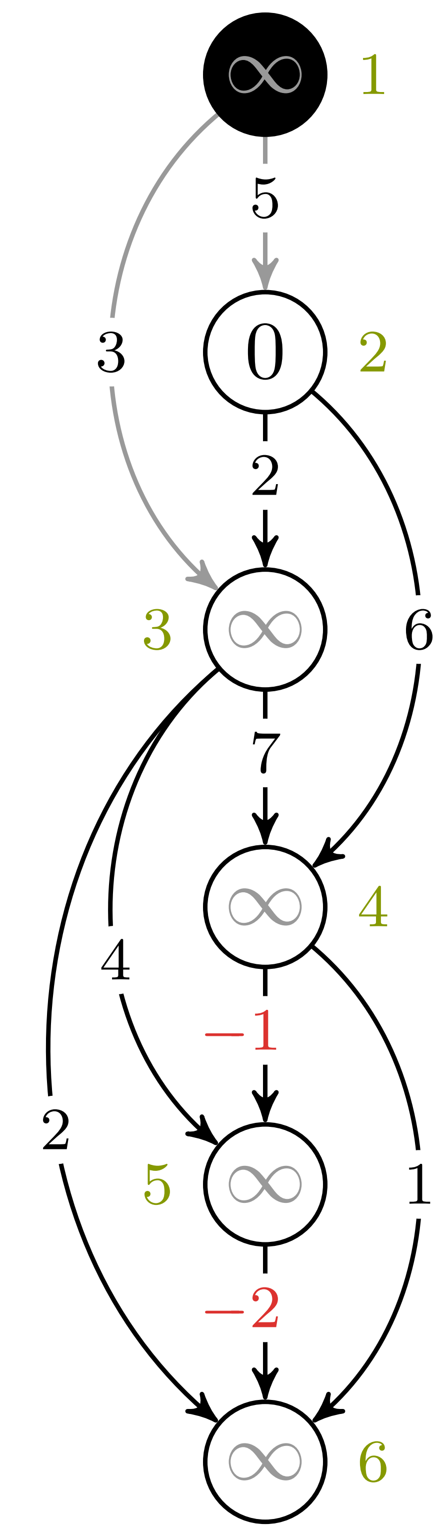
$s, u, v \;=\; 2, 1, 3$






```
DAG-SHORTEST-PATH(G, w, s)
  topologically sort the vertices of G
  INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex u, in topsort order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

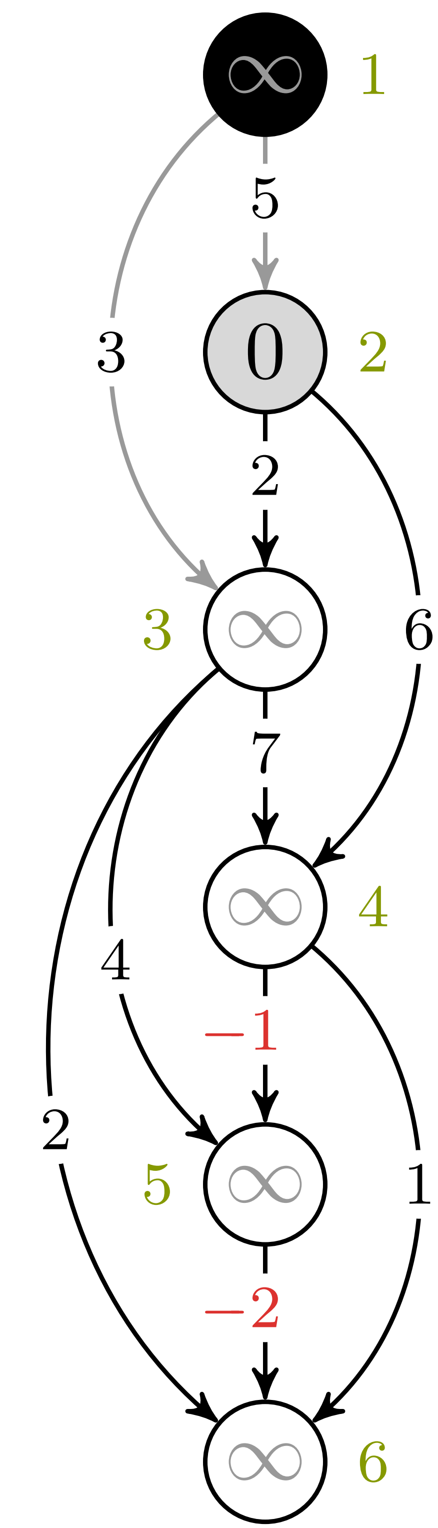
$s, u, v \;=\; 2, 1, 2$



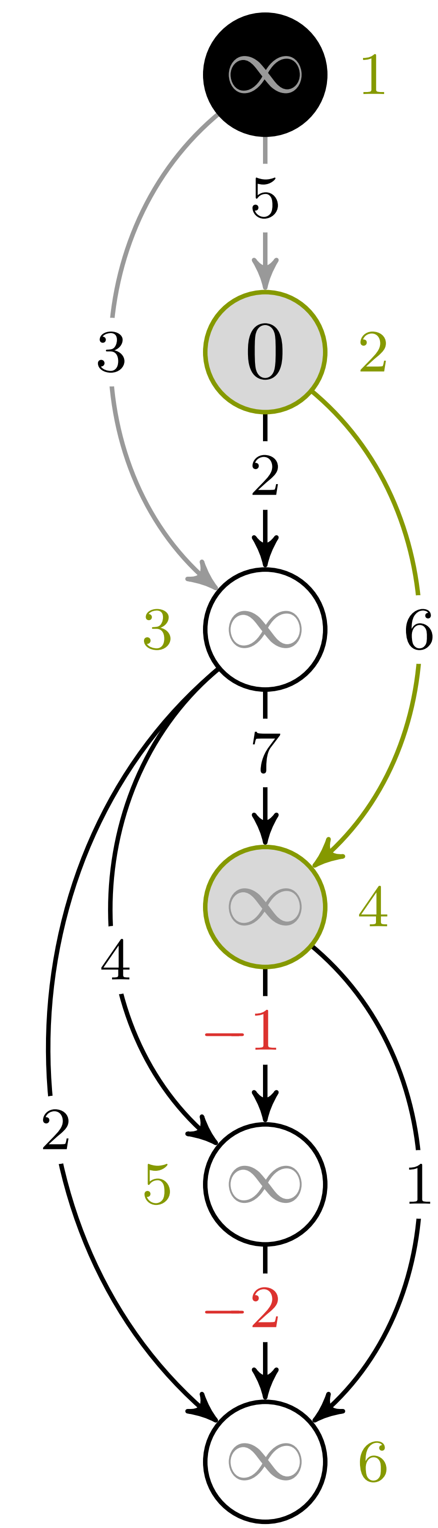

```
DAG-Shortest-Path(G, w, s)
topologically sort the vertices of G
Initialize-Single-Source(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```

$s, u, v \;=\; 2, 1,$ -

```
Dag-Shortest-Path(G, w, s)
topologically sort the vertices of G
Initialize-Single-Source(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```



$s, u, v \;=\; 2, 2, \text{-}$

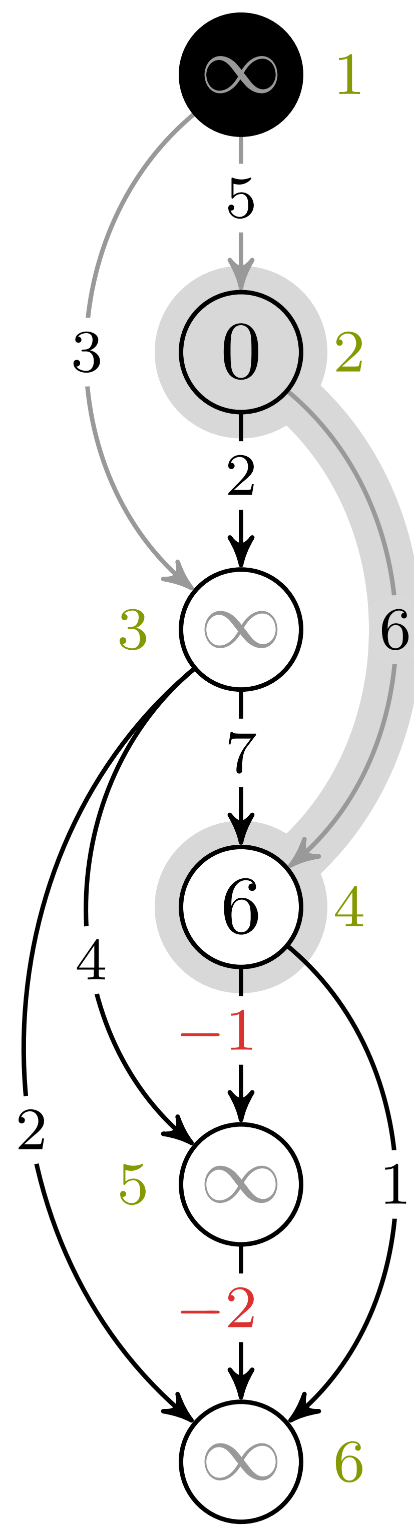



```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$s, u, v \;=\; 2, 2, 4$

```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

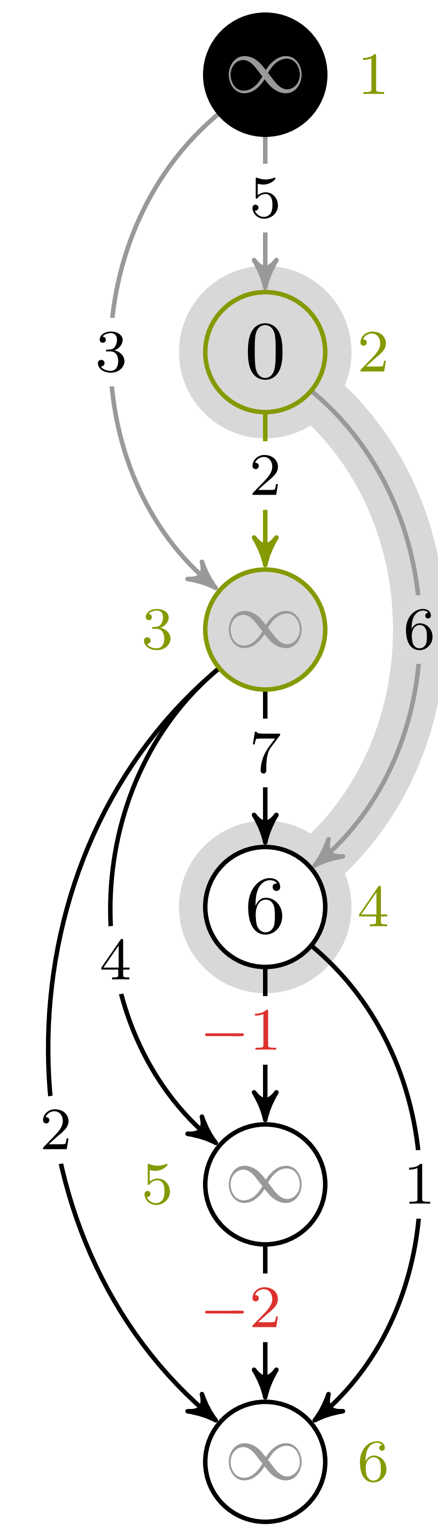
$s, u, v \; = \; 2, 2, 4$



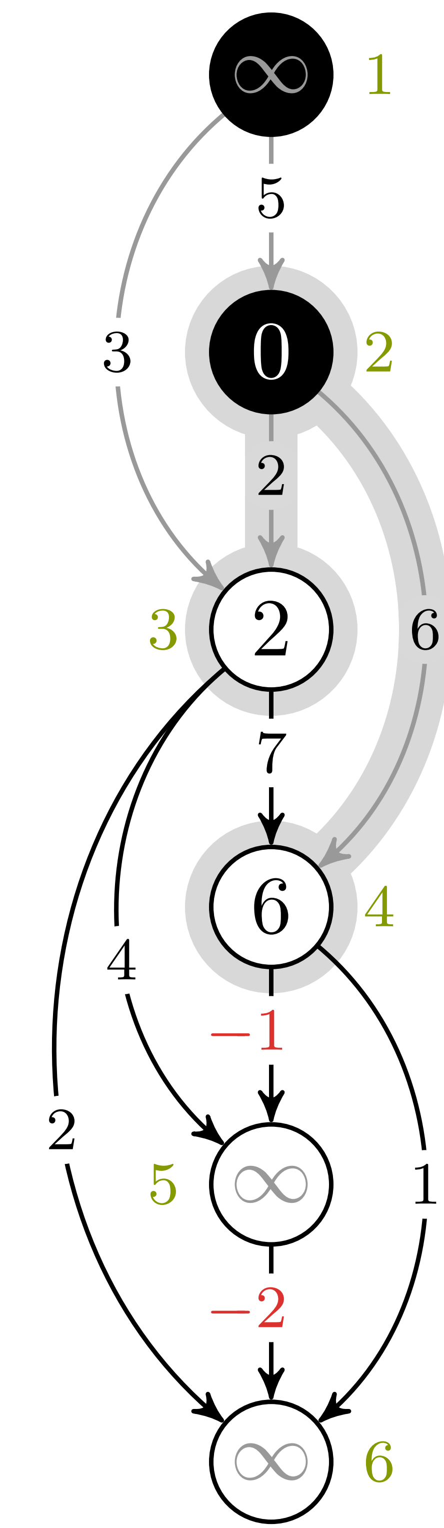
DAG-SHORTEST-PATH(G, w, s)

```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$s, u, v \;=\; 2, 2, 3$

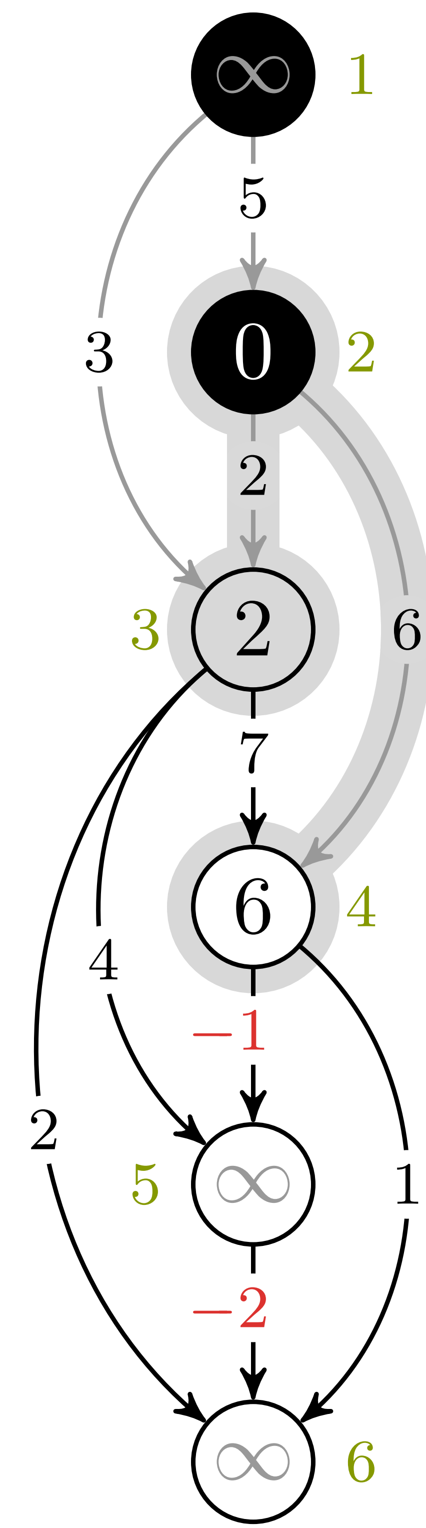


DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
  for each vertex $v \in$ G.$Adj[u]$
    RELAX(u, v, w)



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

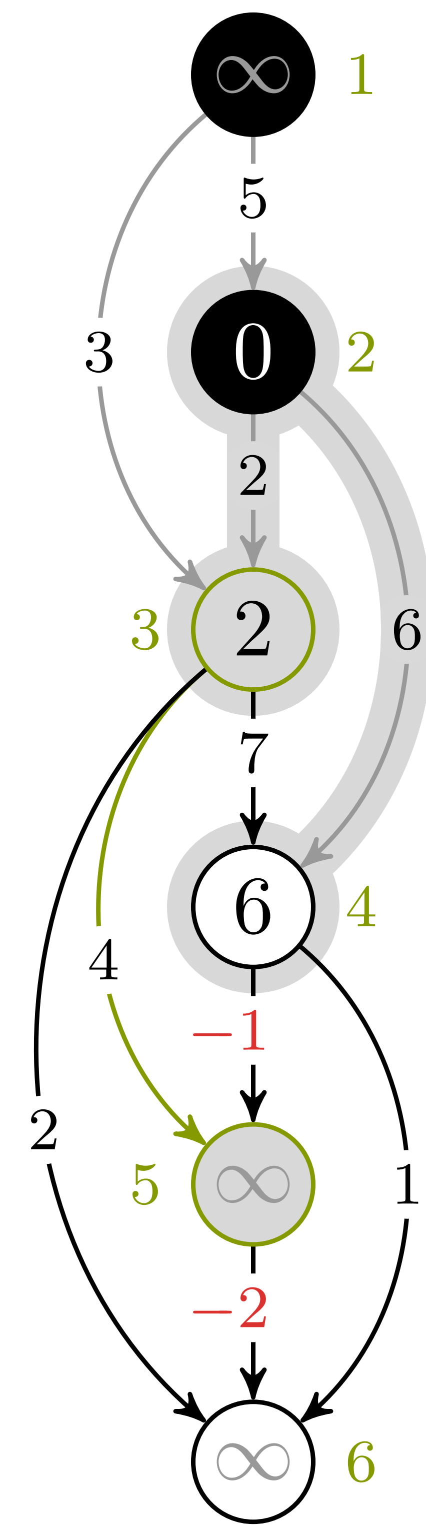
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w)



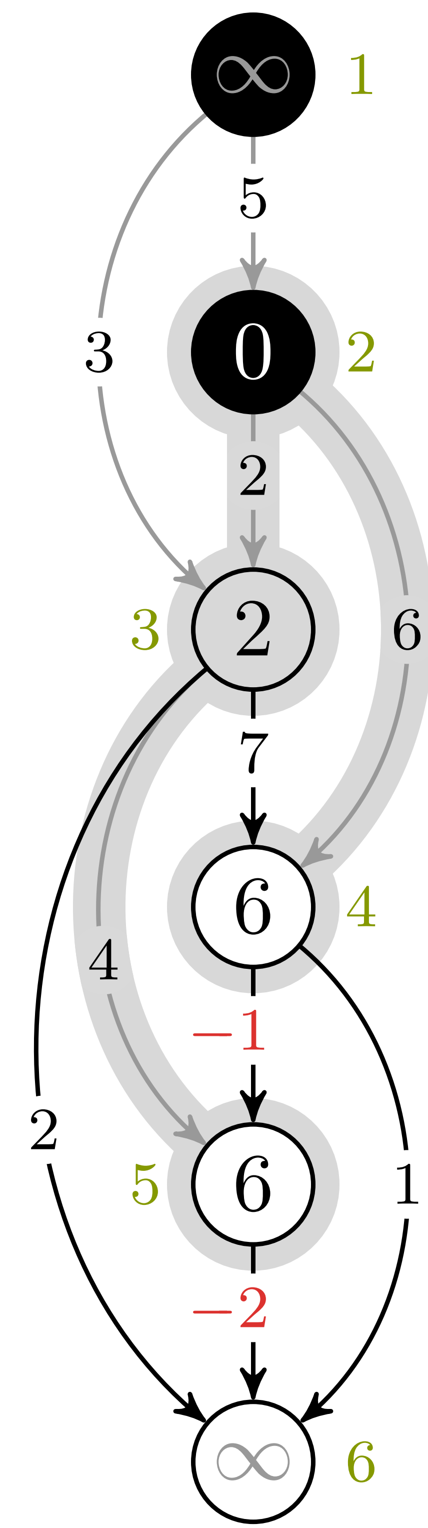
$s, u, v \ = \ 2, 3,$ -

```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$s, u, v \;=\; 2, 3, 5$



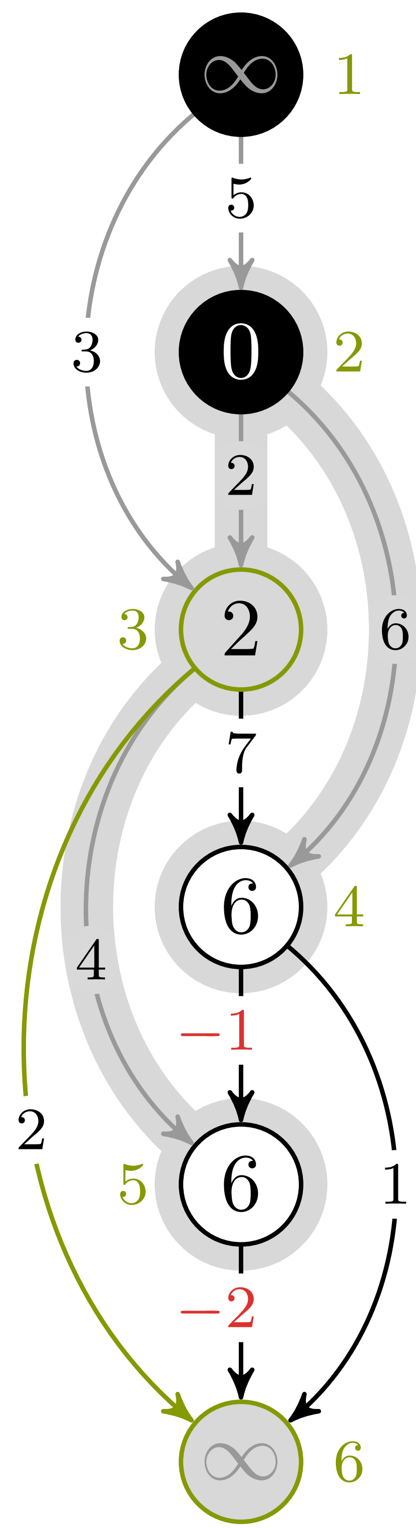
```
DAG-SHORTEST-PATH(G, w, s)
  topologically sort the vertices of G
  INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex u, in topsort order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



$s, u, v \;=\; 2, 3, 5$

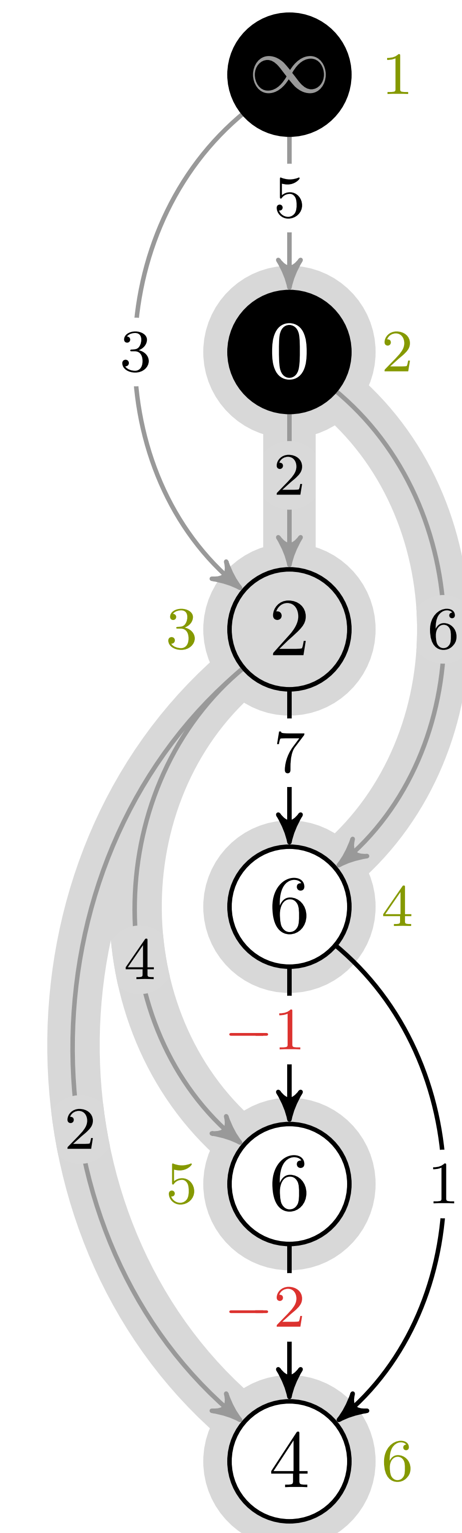
```
DAG-SHORTEST-PATH(G, w, s)
  topologically sort the vertices of G
  INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex u, in topsort order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$s, u, v \;=\; 2, 3, 6$



DAG-SHORTEST-PATH(G, w, s)

```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

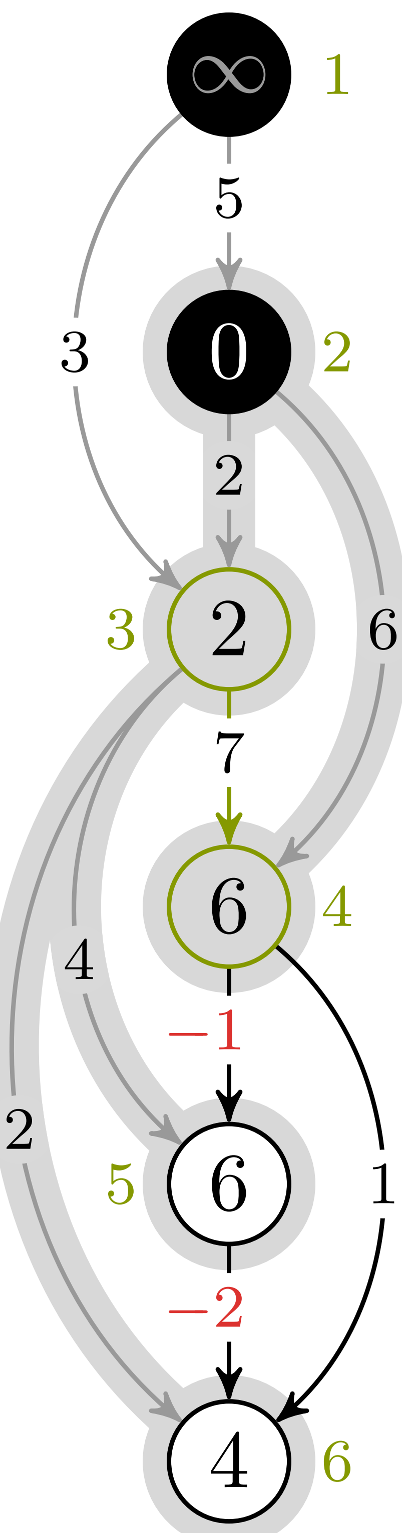


$s, u, v \;=\; 2, 3, 6$

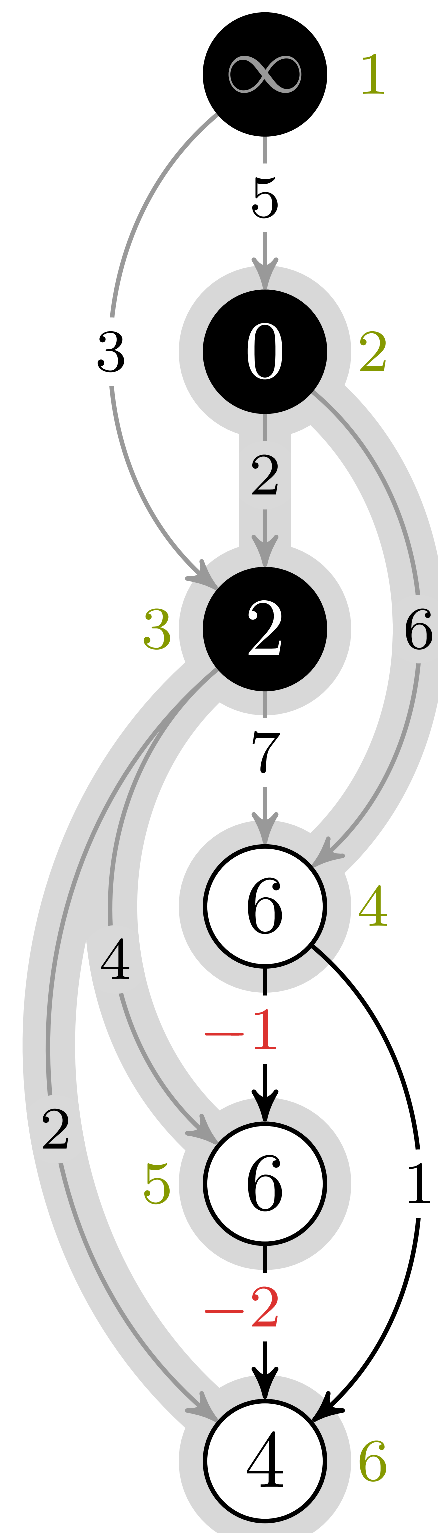
Dag-Shortest-Path(G, w, s)

```
topologically sort the vertices of G
Initialize-Single-Source(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```

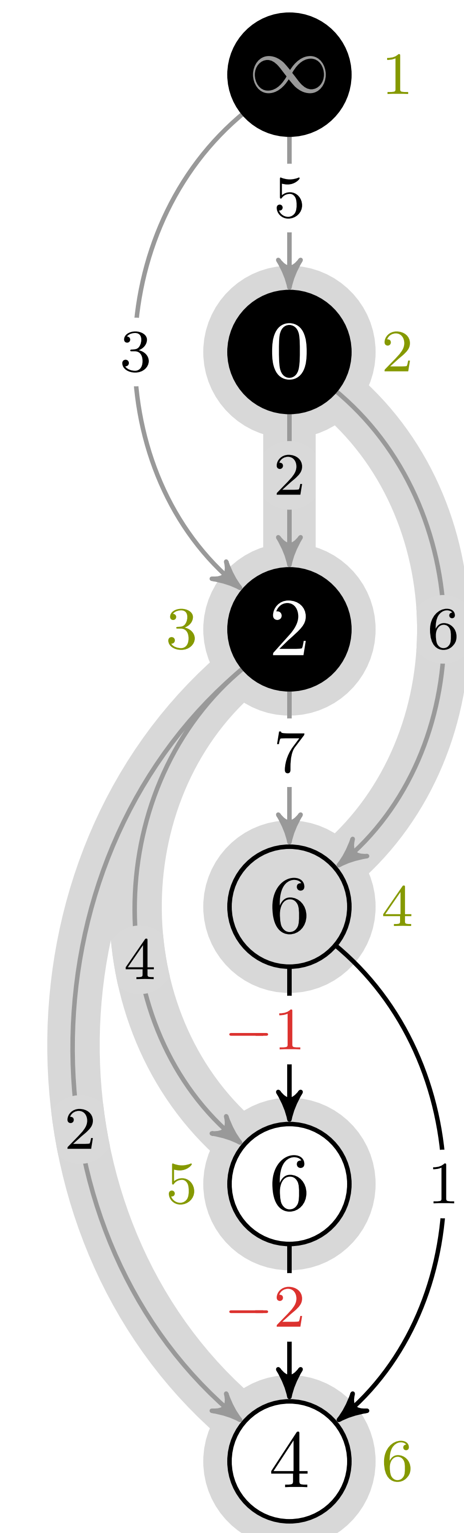
$s, u, v \;=\; 2, 3, 4$



DAG-SHORTEST-PATH(G, w, s)
1 topologically sort the vertices of G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **for** each vertex u, in topsort order
4 **for** each vertex $v \in \mathrm{G}.Adj[u]$
5 RELAX(u, v, w)



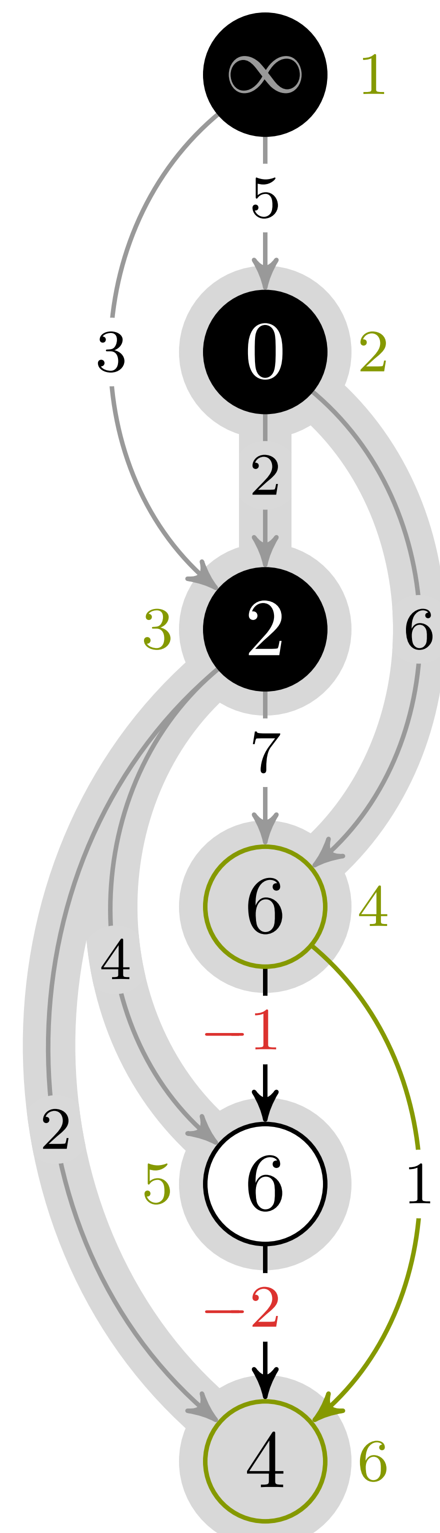
$s, u, v \;=\; 2, 3, \text{-}$




```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$s, u, v \;=\; 2, 4, \text{-}$

```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

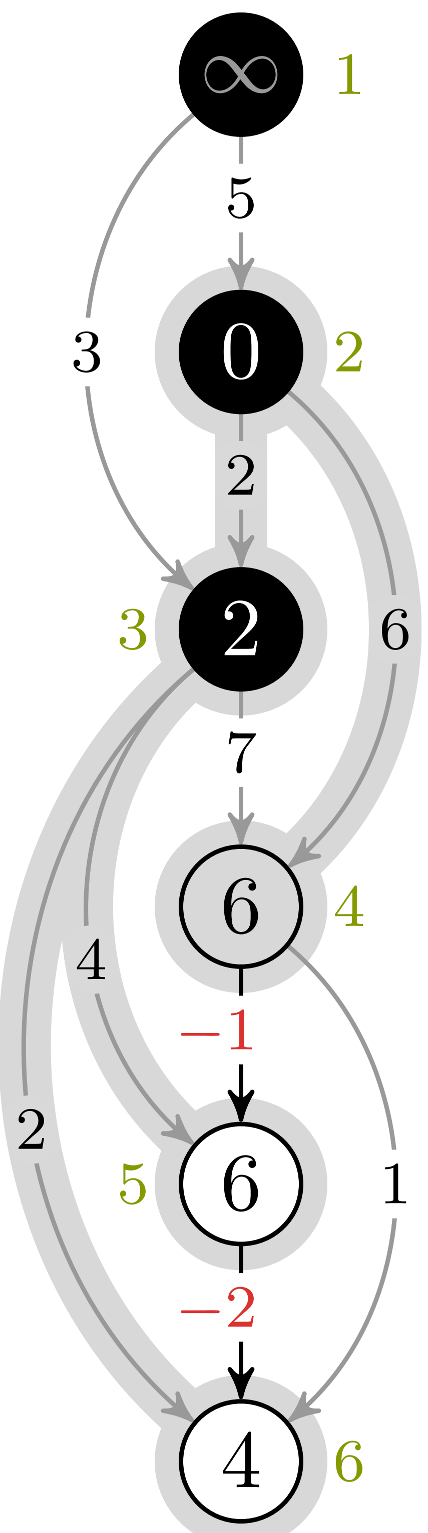


$s, u, v \;=\; 2, 4, 6$

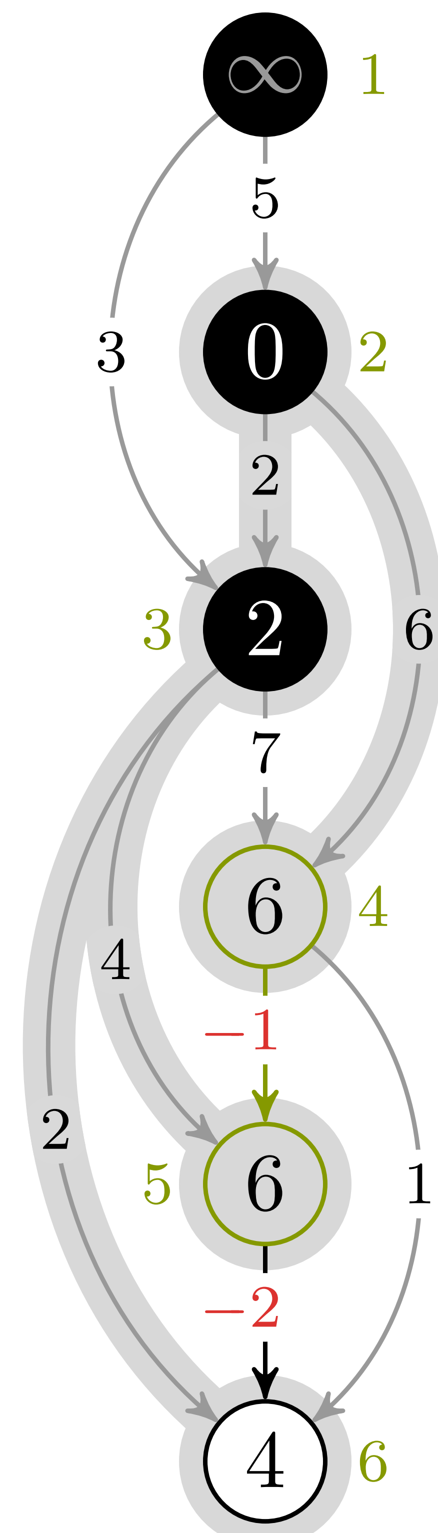
DAG-SHORTEST-PATH(G, w, s)

```
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, in topsort order
4      for each vertex v ∈ G.Adj[u]
5          RELAX(u, v, w)
```

$s, u, v \ = \ 2, 4, 6$



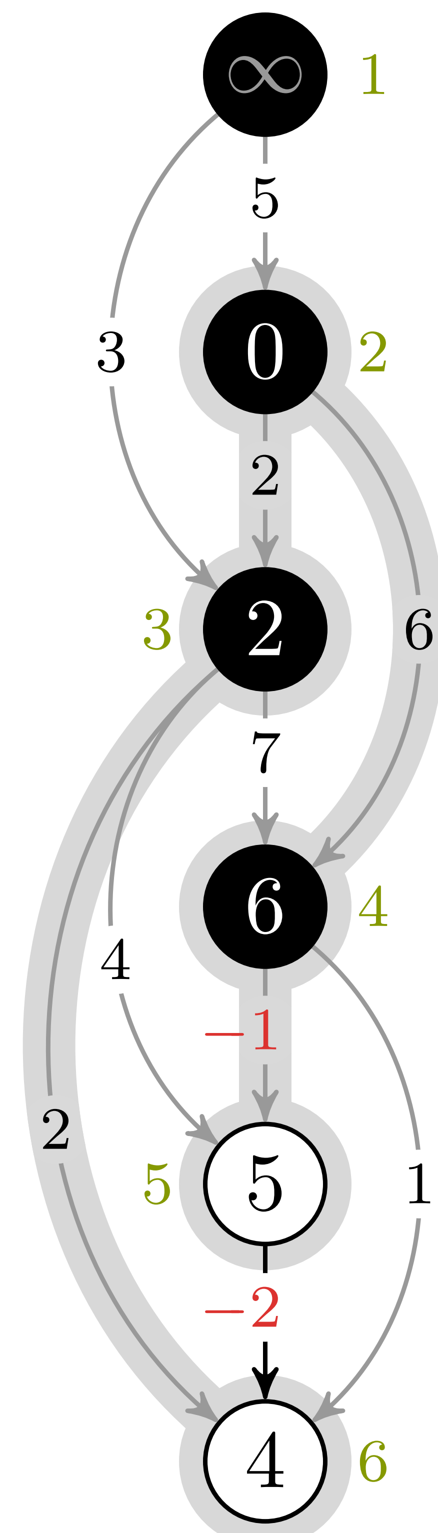
```
DAG-SHORTEST-PATH(G, w, s)
  topologically sort the vertices of G
  INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex u, in topsort order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



$s, u, v \;=\; 2, 4, 5$

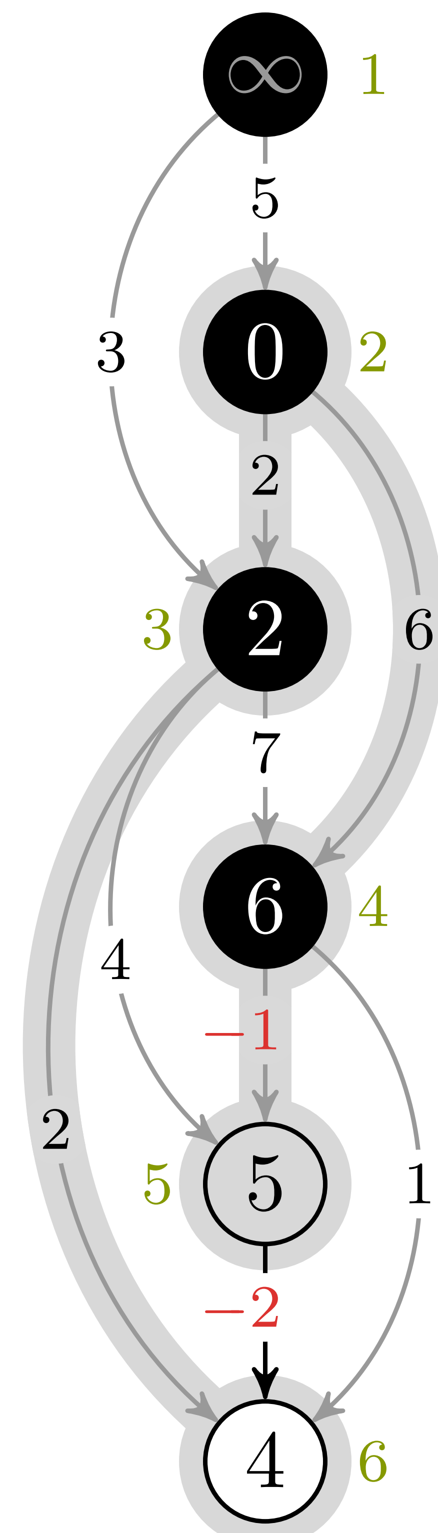
```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$s, u, v \;=\; 2, 4, \text{-}$



DAG-SHORTEST-PATH(G, w, s)

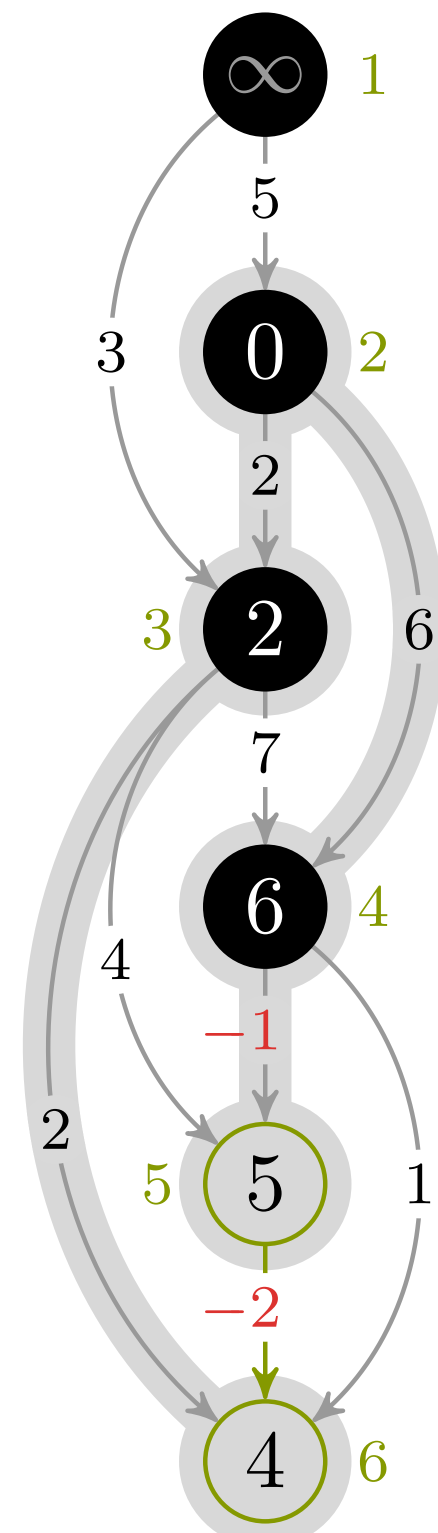
```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



$s, u, v \ = \ 2, 5, \text{-}$

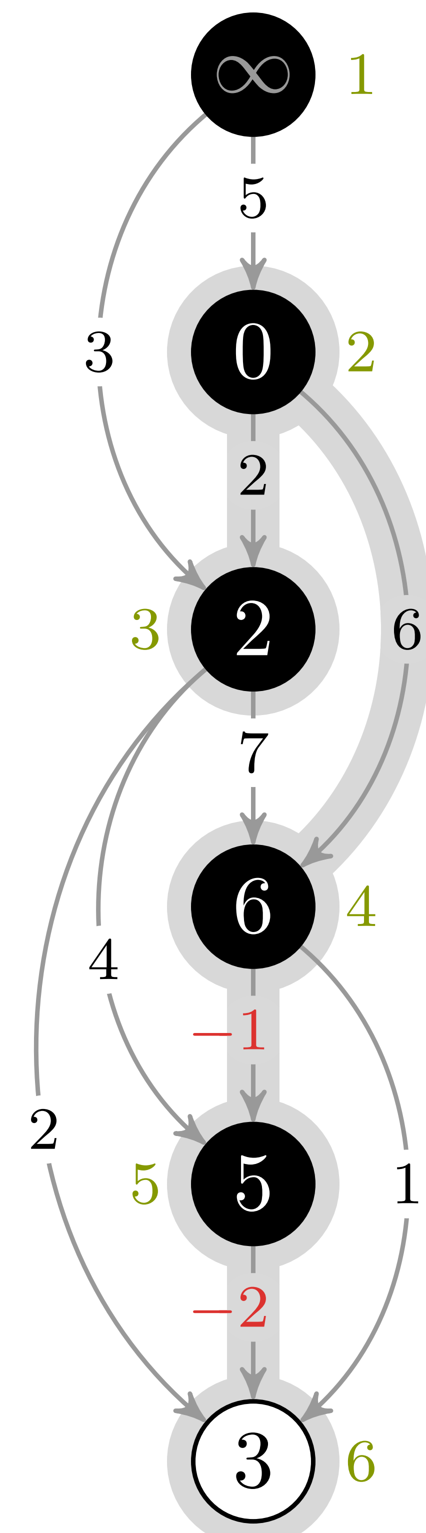
DAG-SHORTEST-PATH(G, w, s)
```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$s, u, v \;=\; 2, 5, 6$



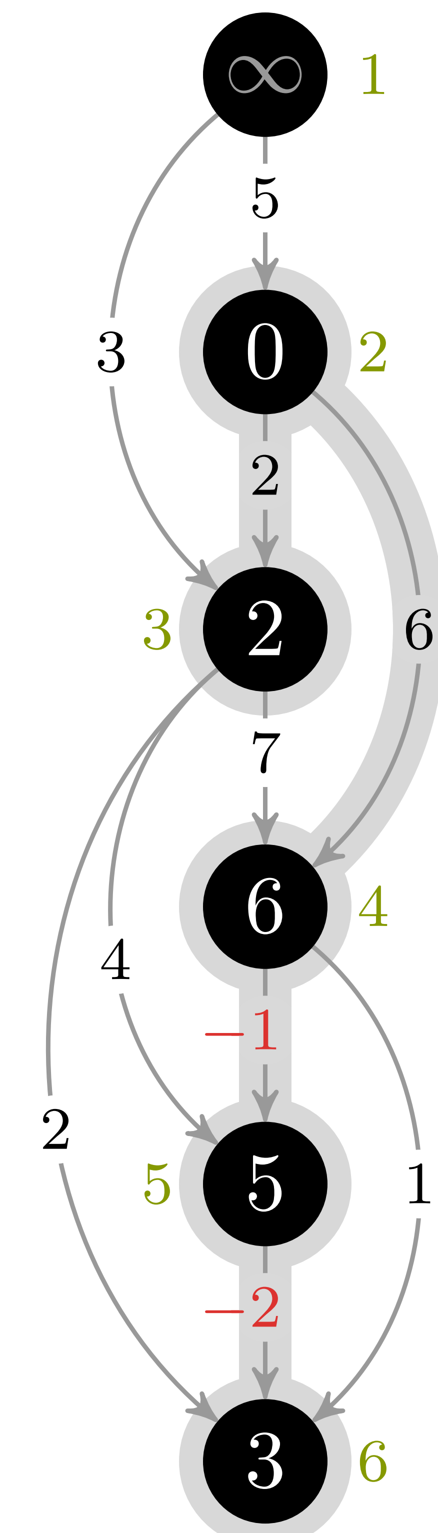
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w)

$s, u, v \;=\; 2, 5,$ -

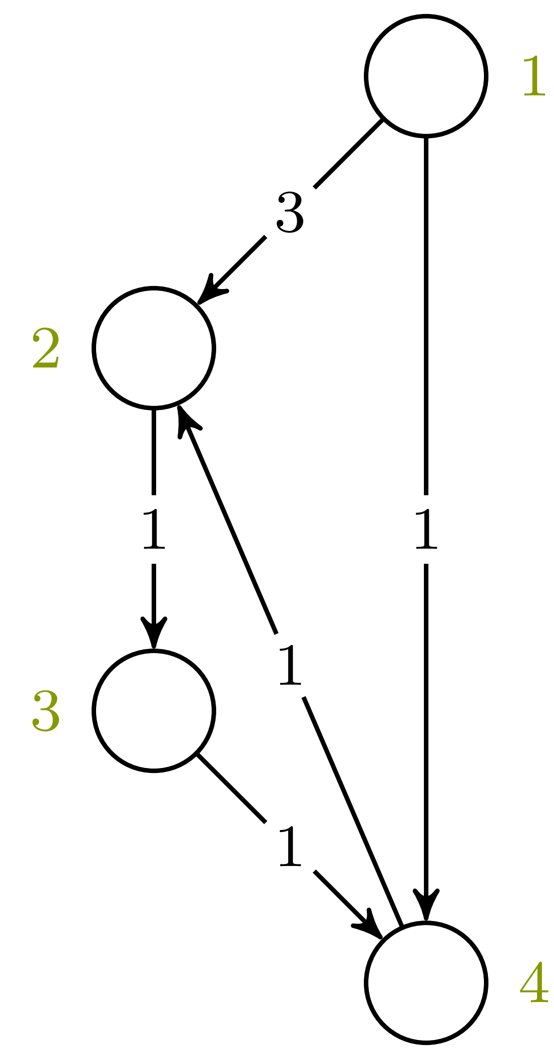


```
DAG-Shortest-Path(G, w, s)
topologically sort the vertices of G
Initialize-Single-Source(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```

$s, u, v \;=\; 2, \text{-}, \text{-}$



```
DAG-SHORTEST-PATH(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
        for each vertex v ∈ G.Adj[u]
                RELAX(u, v, w)
```



$s, u, v \;=\; 1, \text{-}, \text{-}$

DAG-SHORTEST-PATH(G, w, s)

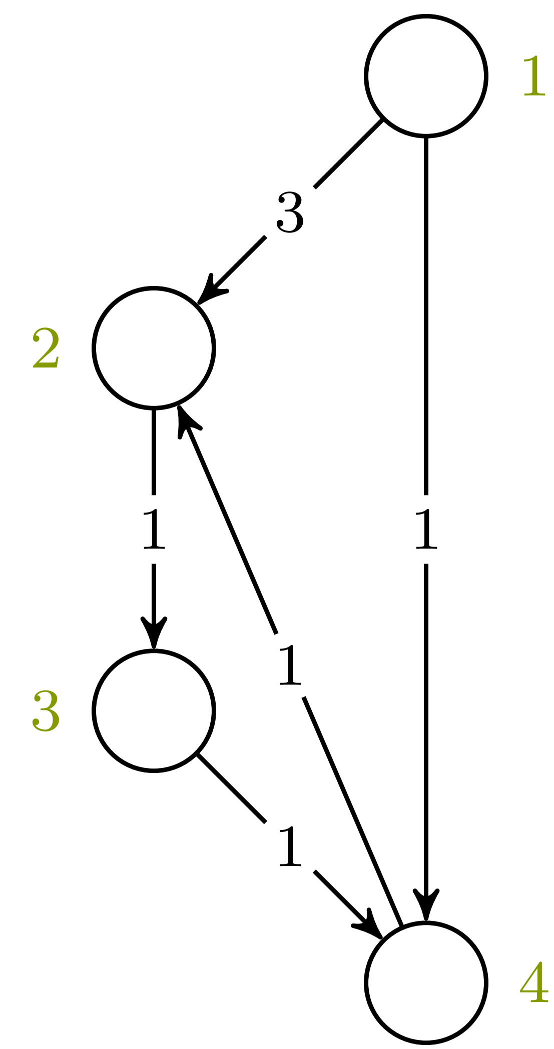
~~topologically sort the vertices of G~~

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex u, in topsort order

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

    RELAX(u, v, w)



$s, u, v \;=\; 1, \text{-}, \text{-}$

DAG-SHORTEST-PATH(G, w, s)

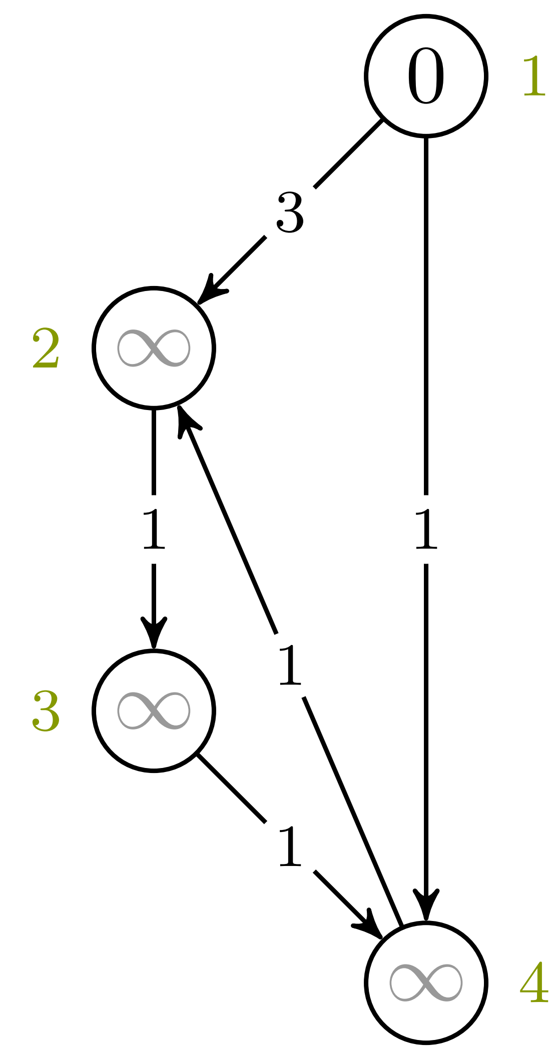
~~topologically sort the vertices of G~~

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex u, in topsort order

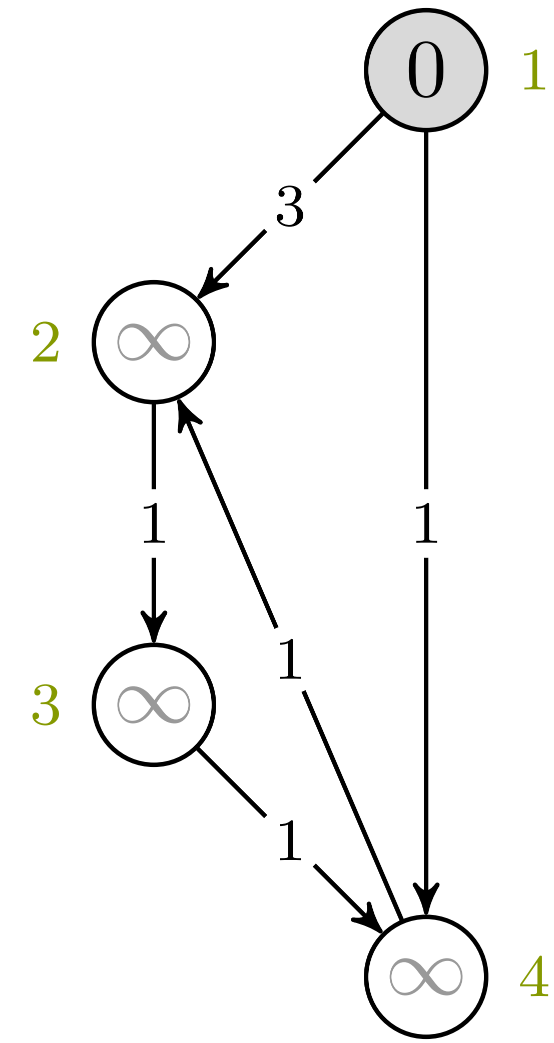
  for each vertex $v \in$ G.$Adj[u]$

    RELAX(u, v, w)



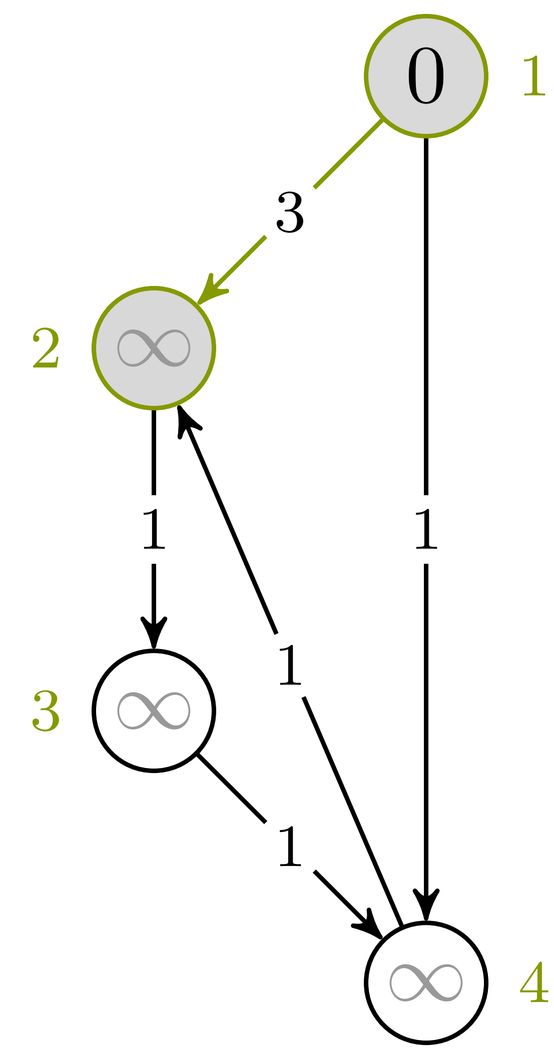
$s, u, v \;=\; 1, \text{-}, \text{-}$

DAG-SHORTEST-PATH(G, w, s)
~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex $v \in$ G.$Adj[u]$
        RELAX(u, v, w)



$s, u, v \;=\; 1, 1, \text{-}$

DAG-SHORTEST-PATH(G, w, s)
~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
  for each vertex $v \in$ G.$Adj[u]$
    RELAX(u, v, w)



$s, u, v \;=\; 1, 1, 2$

DAG-SHORTEST-PATH(G, w, s)

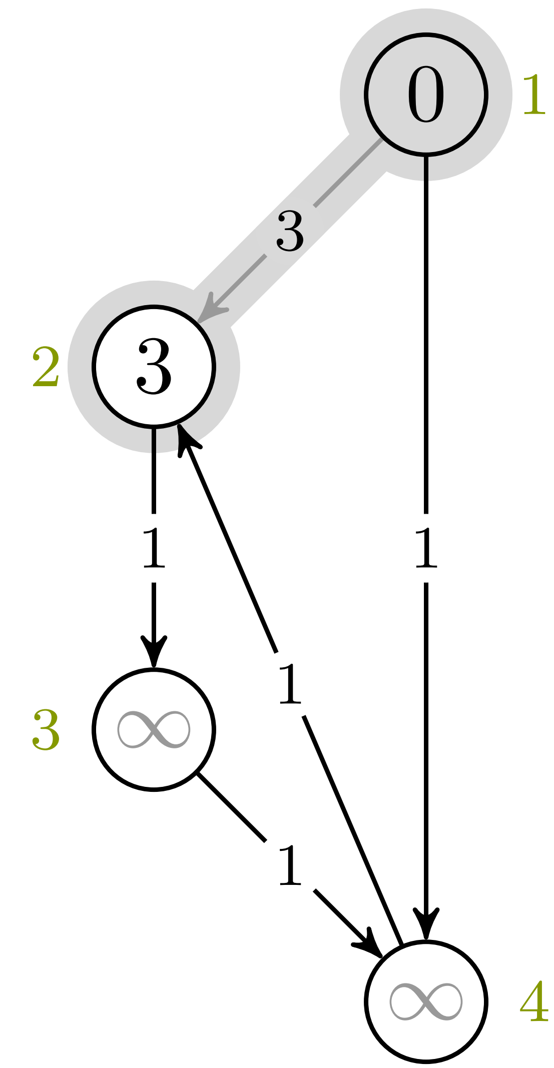
~~topologically sort the vertices of G~~

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex u, in topsort order

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

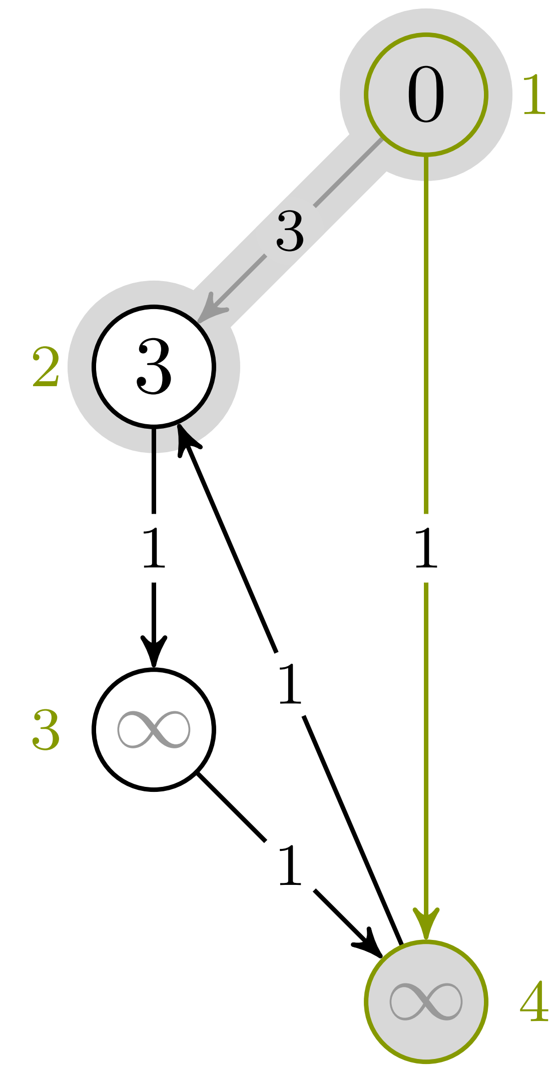
    RELAX(u, v, w)



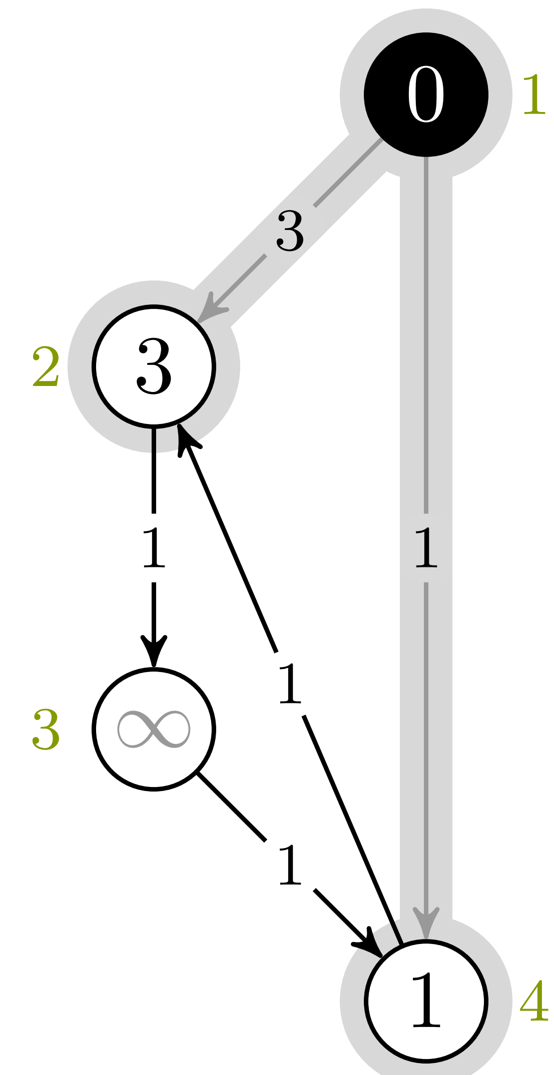
$s, u, v \;=\; 1, 1, 2$

```
DAG-SHORTEST-PATH(G, w, s)
~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$s, u, v \;=\; 1, 1, 4$



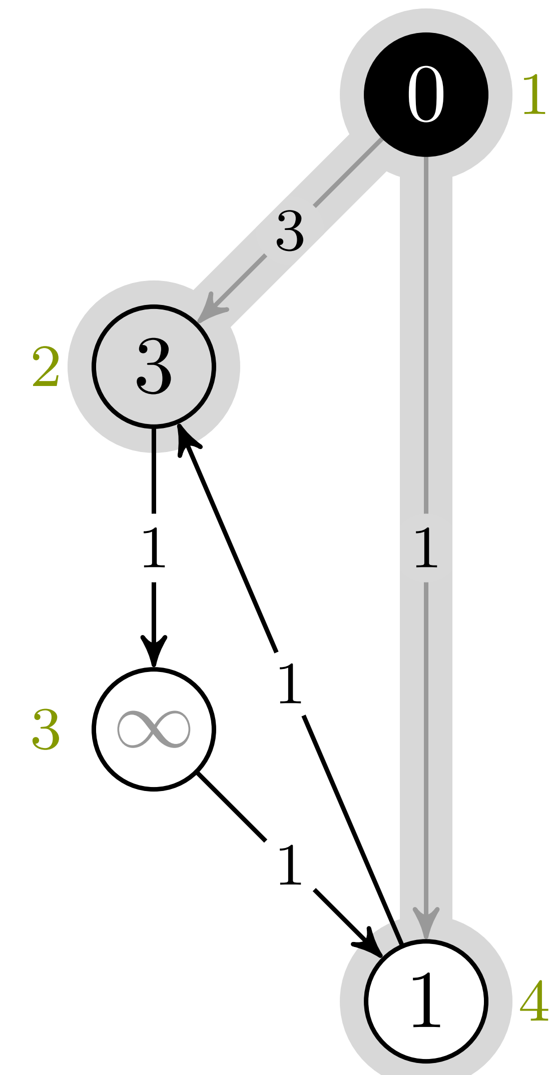
```
DAG-SHORTEST-PATH(G, w, s)
~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



$s, u, v \;=\; 1, 1, \text{-}$

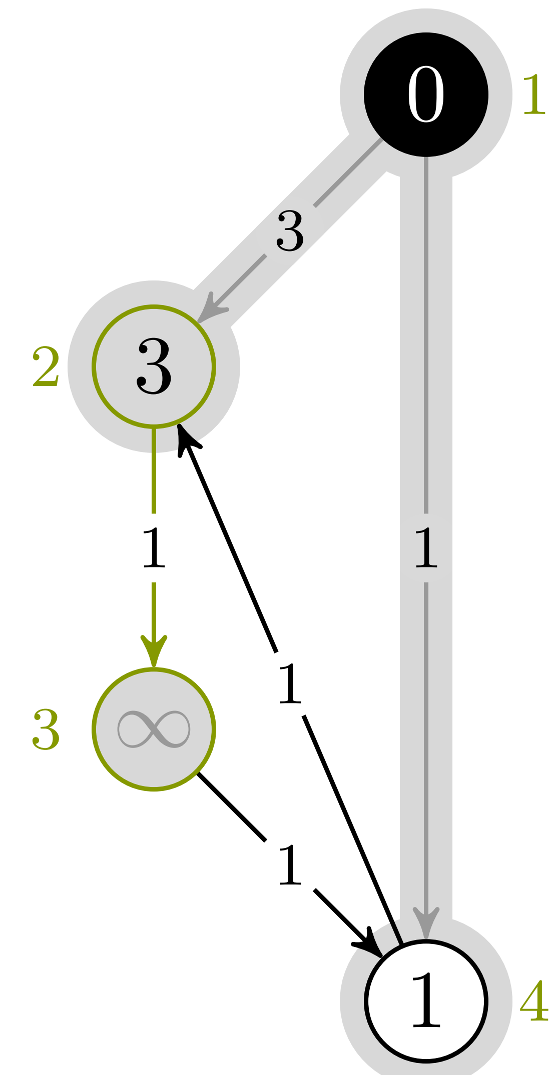
DAG-SHORTEST-PATH(G, w, s)

~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex $v \in$ G.$Adj[u]$
        RELAX(u, v, w)



$s, u, v \ = \ 1, 2, -$

```
DAG-SHORTEST-PATH(G, w, s)
~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



$s, u, v \;=\; 1, 2, 3$

DAG-SHORTEST-PATH(G, w, s)

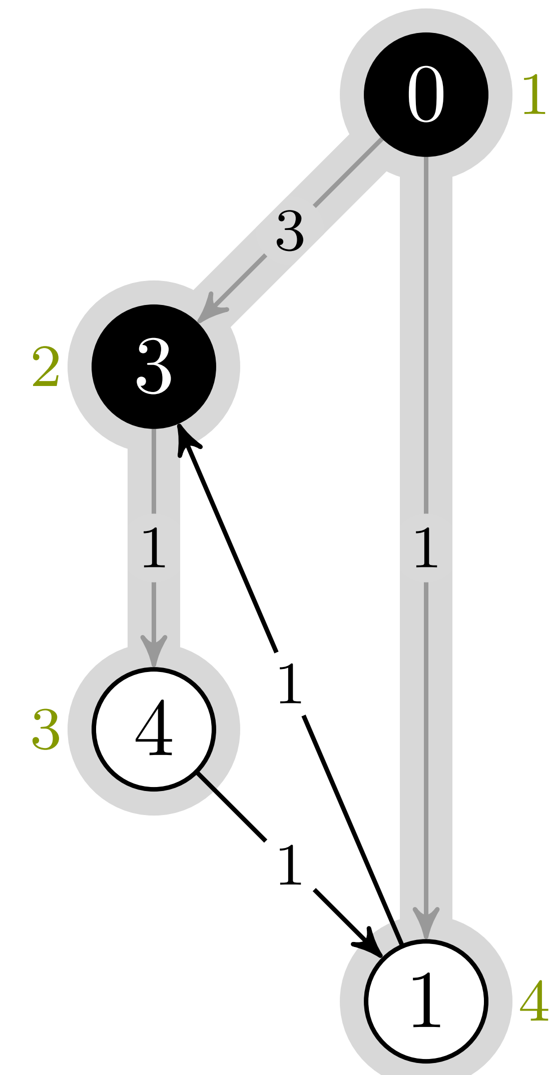
~~topologically sort the vertices of G~~

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex u, in topsort order

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

    RELAX(u, v, w)

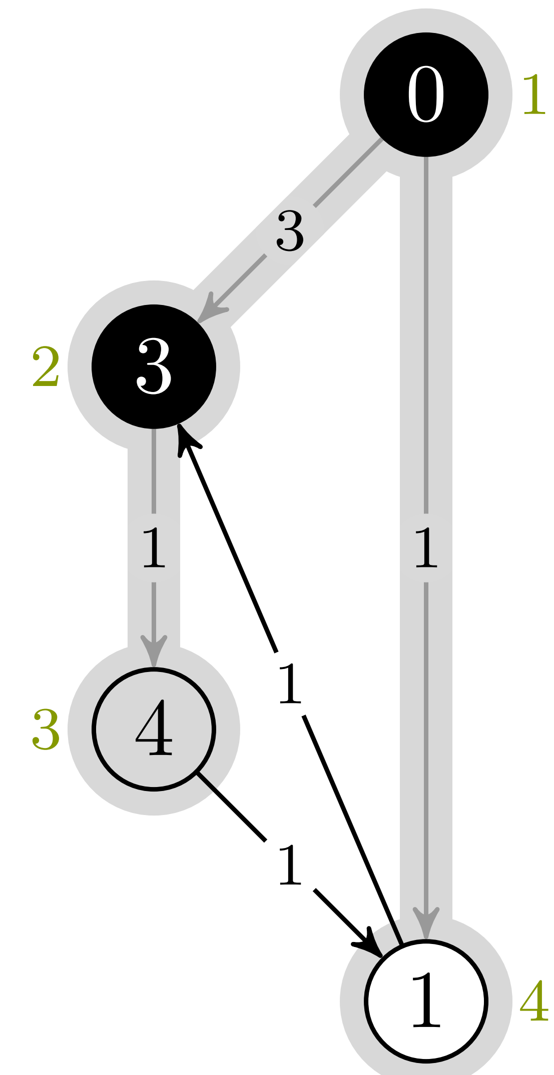


$s, u, v \; = \; 1, 2, -$

DAG-SHORTEST-PATH(G, w, s)

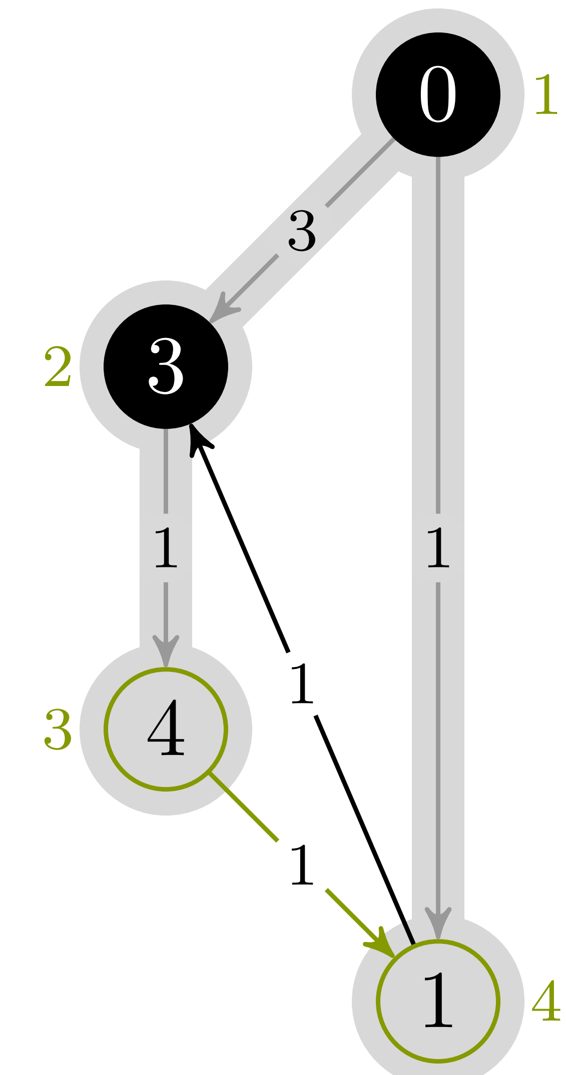
```
~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

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

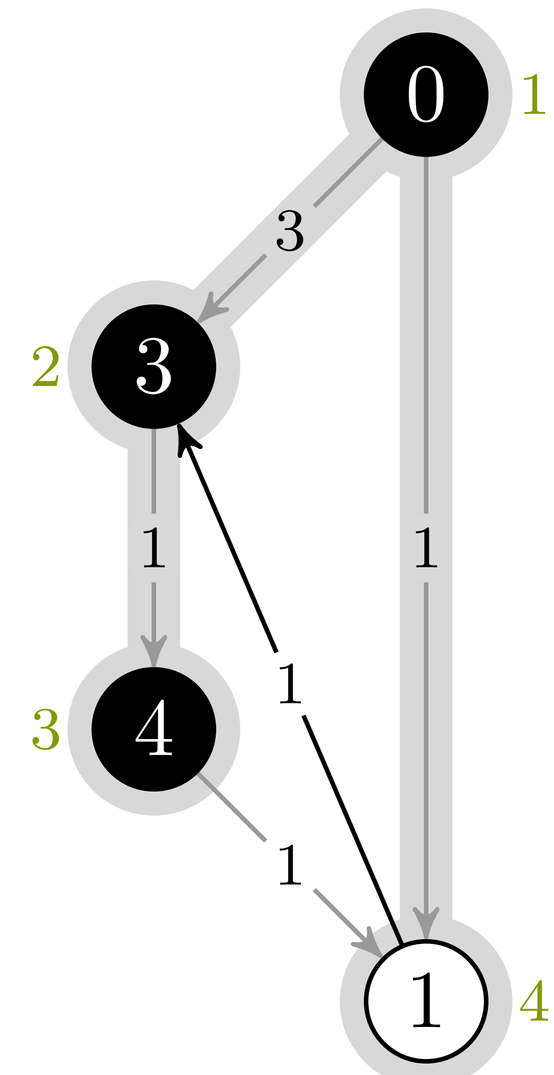


```
DAG-Shortest-Path(G, w, s)
~~topologically sort the vertices of G~~
Initialize-Single-Source(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```

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



```
DAG-SHORTEST-PATH(G, w, s)
~~topologically sort the vertices of G~~
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, in topsort order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



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

DAG-SHORTEST-PATH(G, w, s)

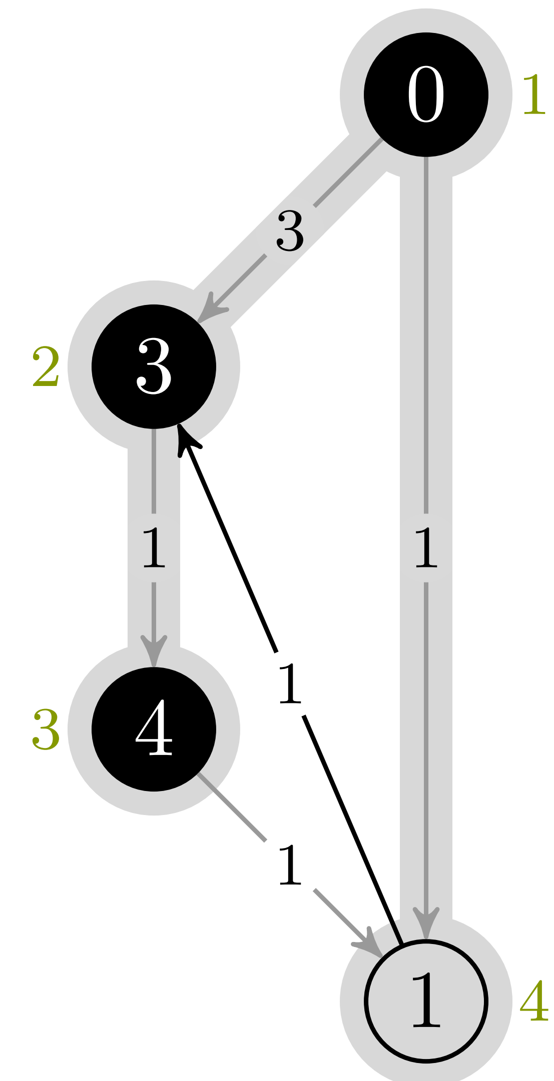
~~topologically sort the vertices of G~~

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex u, in topsort order

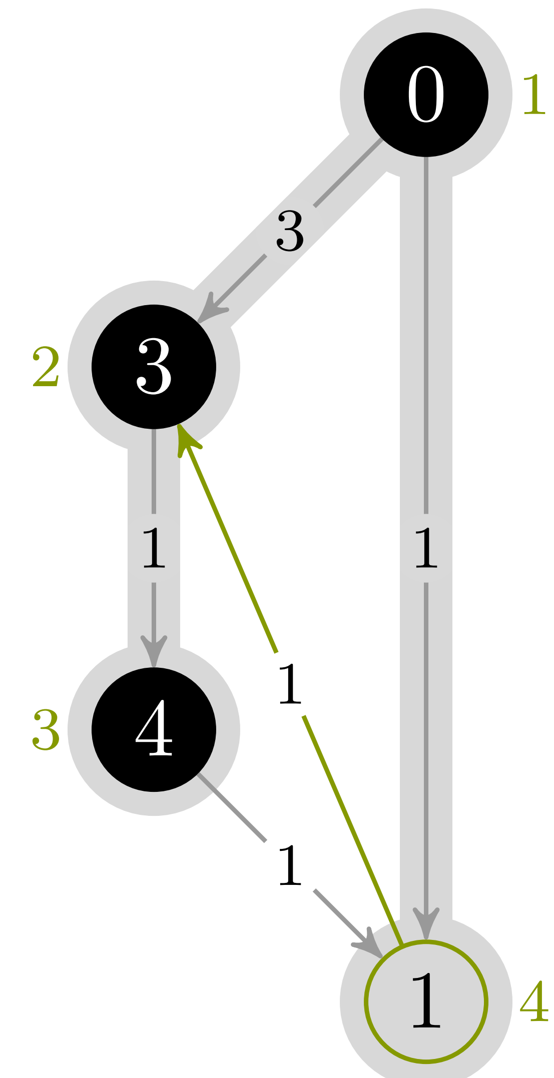
    for each vertex $v \in$ G.$Adj[u]$

        RELAX(u, v, w)



$s, u, v \;=\; 1, 4, \text{–}$

```
DAG-Shortest-Path(G, w, s)
    ~~topologically sort the vertices of G~~
    Initialize-Single-Source(G, s)
    for each vertex u, in topsort order
        for each vertex v ∈ G.Adj[u]
            Relax(u, v, w)
```



$s, u, v \;=\; 1, 4, 2$

DAG-SHORTEST-PATH(G, w, s)

~~topologically sort the vertices of G~~

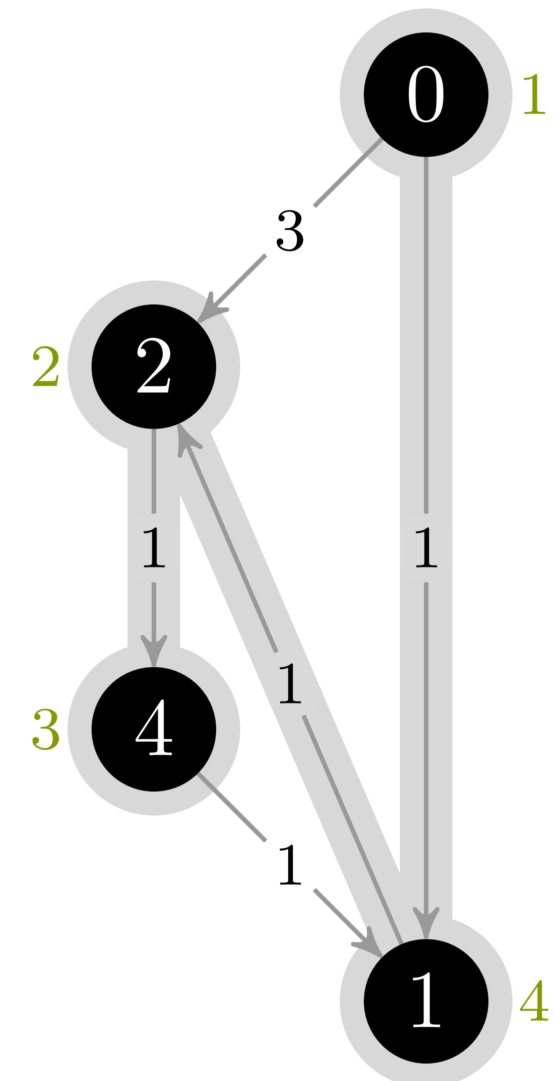
INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex u, in topsort order

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

    RELAX(u, v, w)

$s, u, v \;=\; 1, \text{-}, \text{-}$



$3.d$ er nå feil

DAG-SHORTEST-PATH(… $, s$)

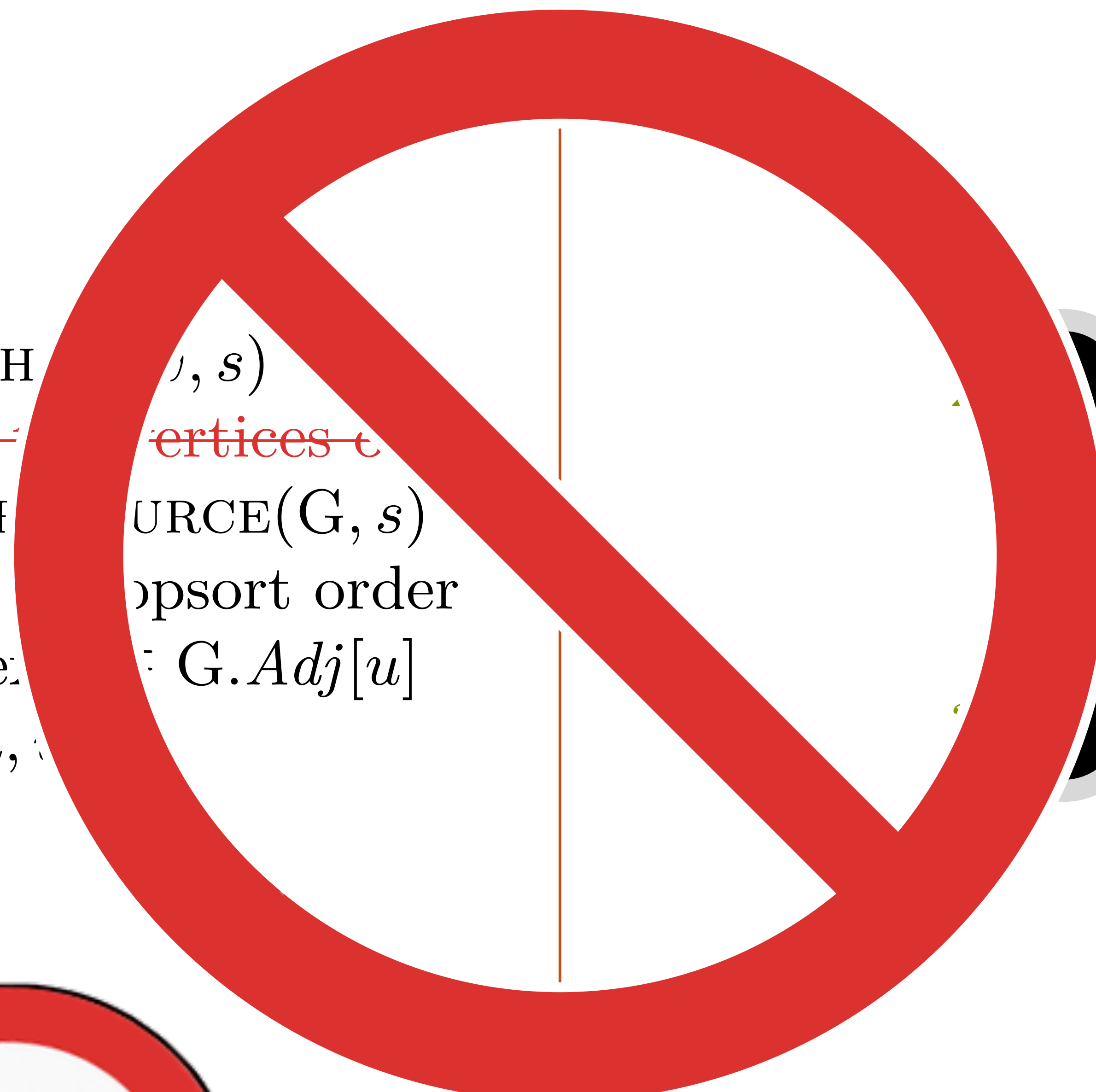
~~topologically sort t~~ … ~~ertices c~~

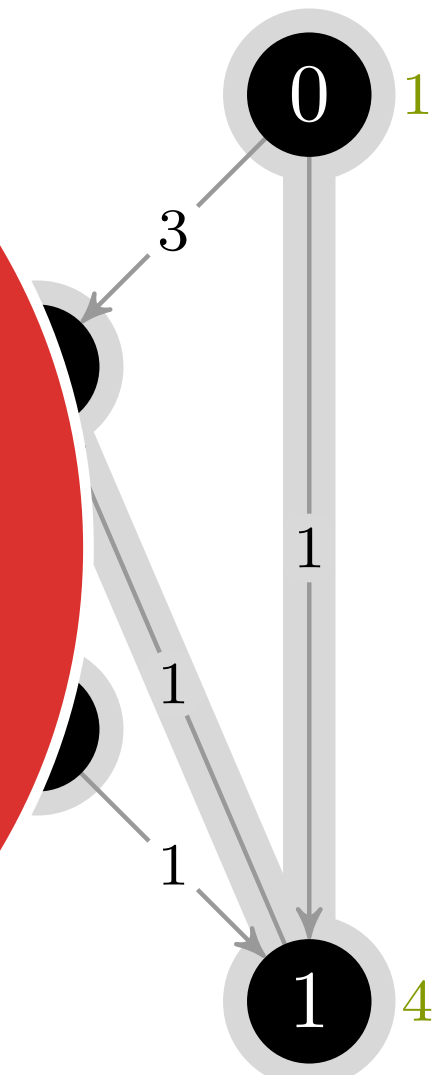
INITIALIZE-SINGLE … URCE(G, s)

for each vertex u, … psort order

  for each verte … $G.Adj[u]$

    RELAX(u, …





Sykler forbudt!

3.d er nå feil

- **God mental modell for dynamisk programmering; erkeeksempel**
- **Delproblemer er avstander fra s til inn-naboer; velg den som gir deg best resultat**
- **Bottom-up: Kantslakking av inn-kanter i topologisk sortert rekkefølge (såkalt pulling)**
- **Gir samme svar: Kantslakking av ut-kanter i topologisk sortert rekkefølge (såkalt reaching)**

DAG-SP › Kjøretid

Operasjon	Antall	Kjøretid

Operasjon	Antall	Kjøretid
Topologisk sortering		

Operasjon	**Antall**	**Kjøretid**
Topologisk sortering	1	

Operasjon	Antall	Kjøretid
Topologisk sortering	1	$\Theta(V + E)$

Operasjon	**Antall**	**Kjøretid**
Topologisk sortering	1	$\Theta(V + E)$
Initialisering		

Operasjon	**Antall**	**Kjøretid**
Topologisk sortering	1	$\Theta(V + E)$
Initialisering	1	

Operasjon	Antall	Kjøretid
Topologisk sortering	1	$\Theta(V + E)$
Initialisering	1	$\Theta(V)$

Operasjon	**Antall**	**Kjøretid**
Topologisk sortering	1	$\Theta(V + E)$
Initialisering	1	$\Theta(V)$
Relax		

Operasjon	**Antall**	**Kjøretid**
Topologisk sortering	1	$\Theta(V + E)$
Initialisering	1	$\Theta(V)$
Relax	E	

Operasjon	**Antall**	**Kjøretid**
Topologisk sortering	1	$\Theta(V + E)$
Initialisering	1	$\Theta(V)$
Relax	E	$\Theta(1)$

Operasjon	**Antall**	**Kjøretid**
Topologisk sortering	1	$\Theta(V + E)$
Initialisering	1	$\Theta(V)$
Relax	E	$\Theta(1)$

Totalt: $\Theta(V + E)$

Kantslakking er altså en oppspalting av minimums-operasjonen fra dekomponeringen. Vi har foreløpig ikke vært så kreative med hvordan vi har brukt det – la oss studere teknikken litt mer i detalj.

3:5

Kantslakking

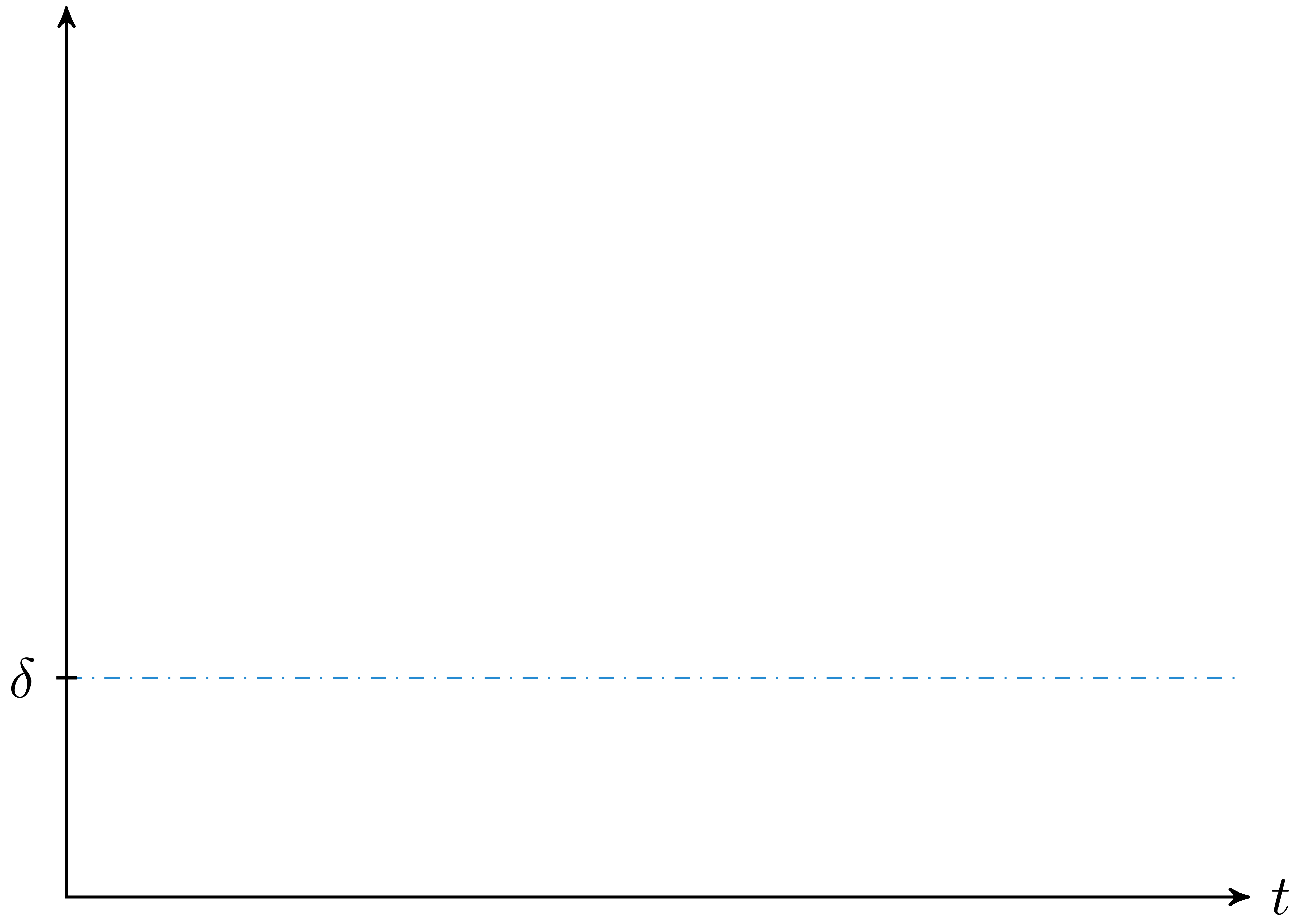
Se fotnote på side 648 i boka for mer om navnet.

$$\delta(s, v) \leqslant v.d$$

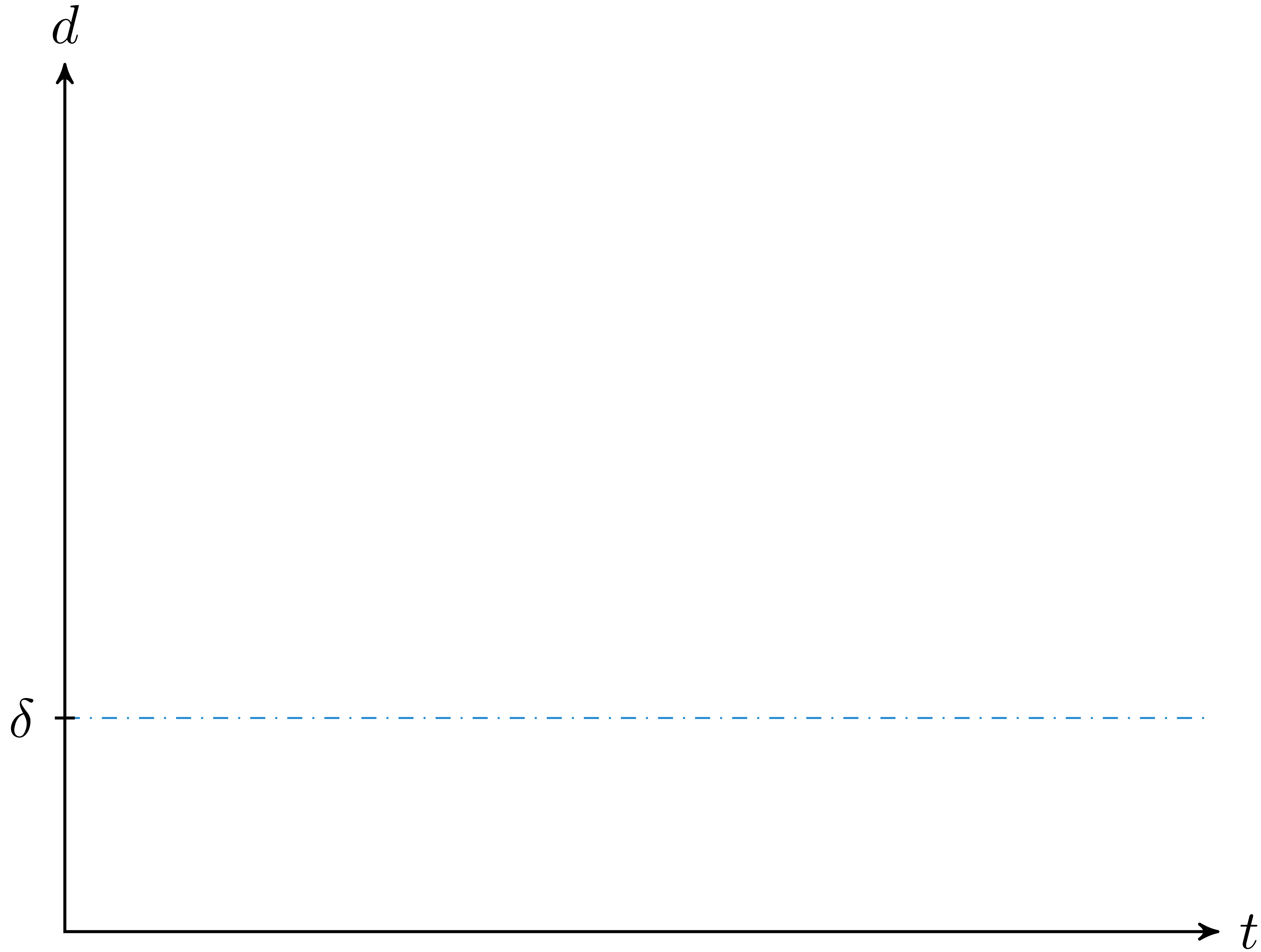
Avstanden fra s til v: Lengden til korteste vei

$$\delta(s, v) \leqslant v.d$$

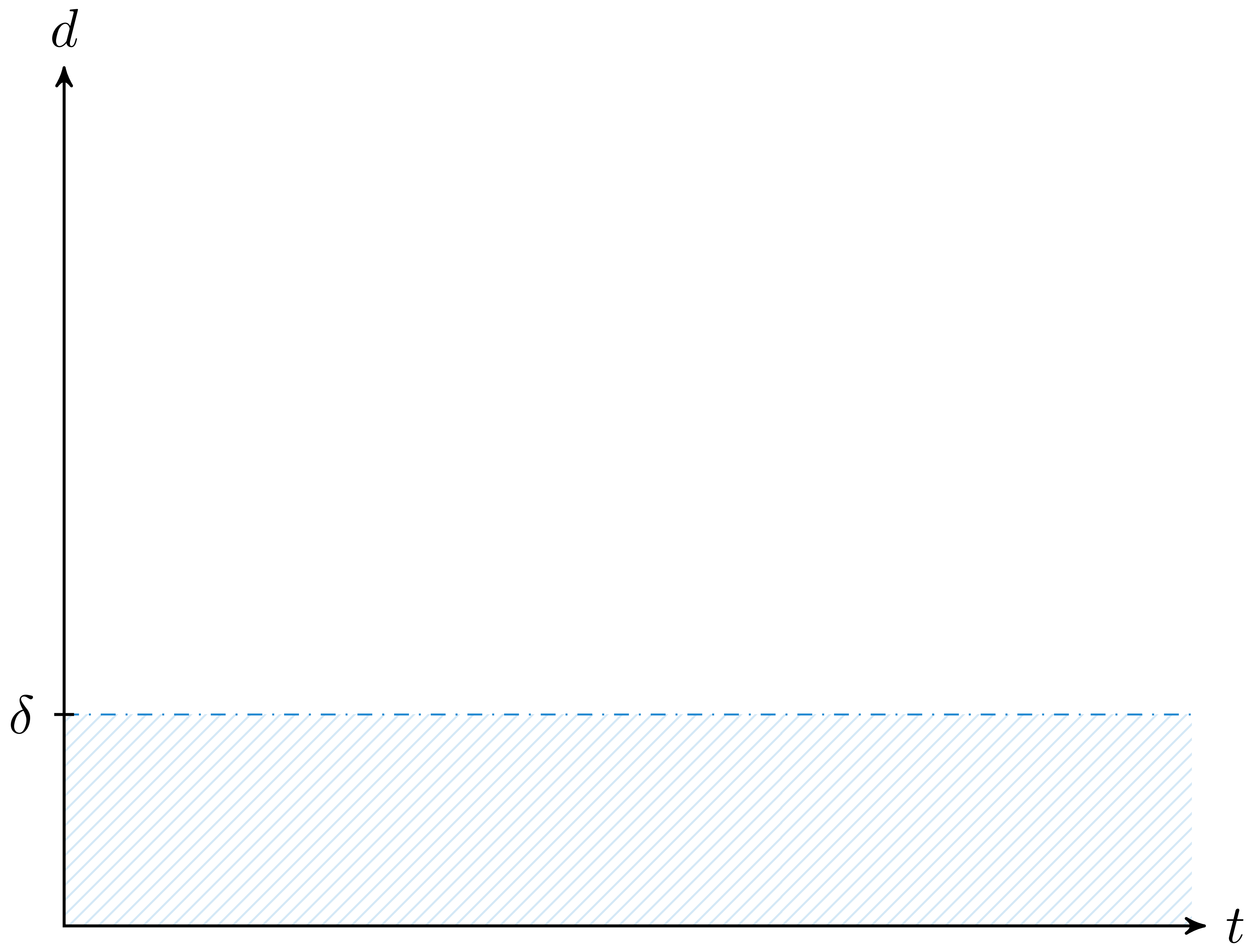
Et overestimat: Det beste vi har funnet så langt!



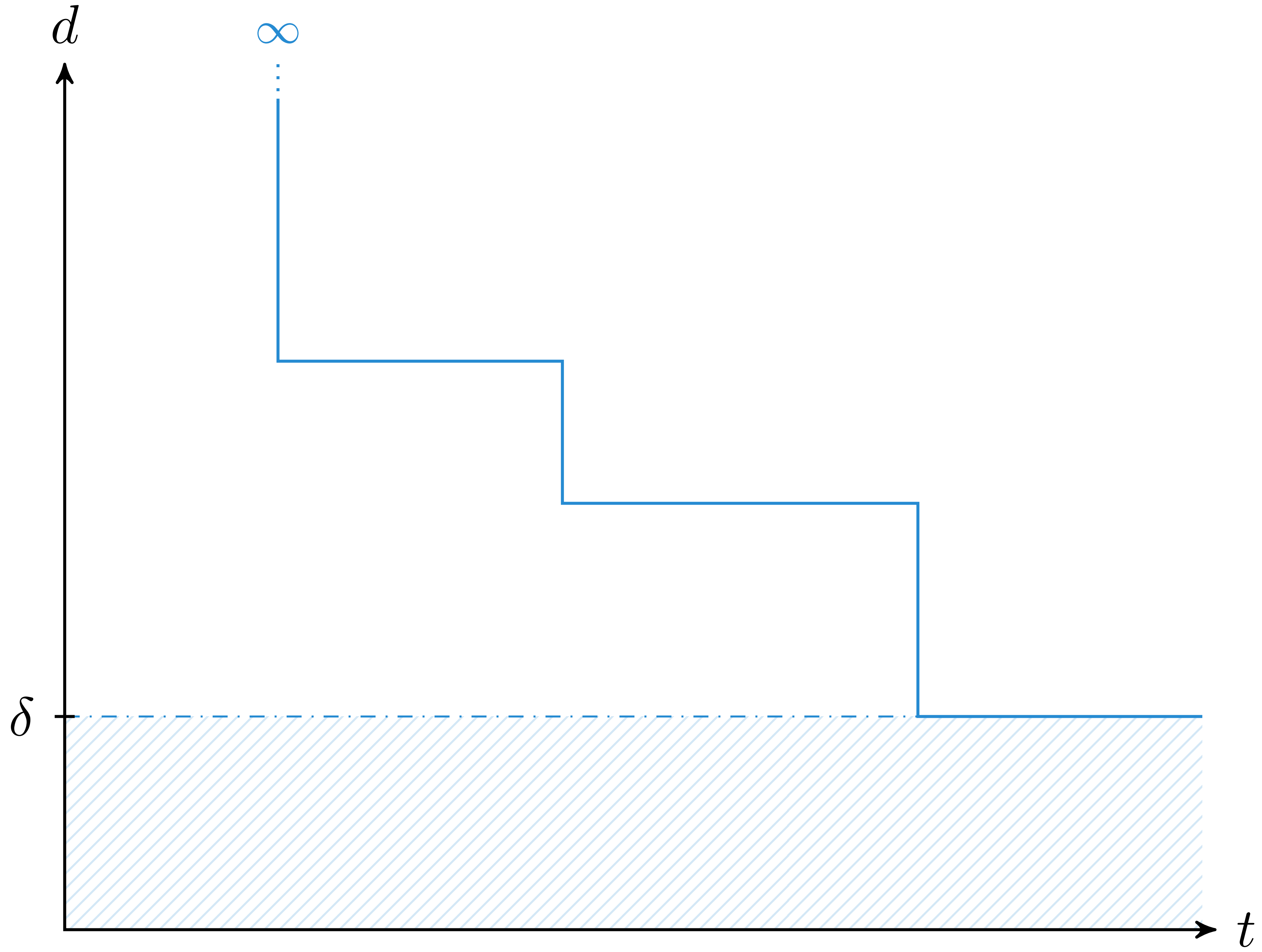
Avstanden $\delta(s, v)$ er ukjent til å begynne med



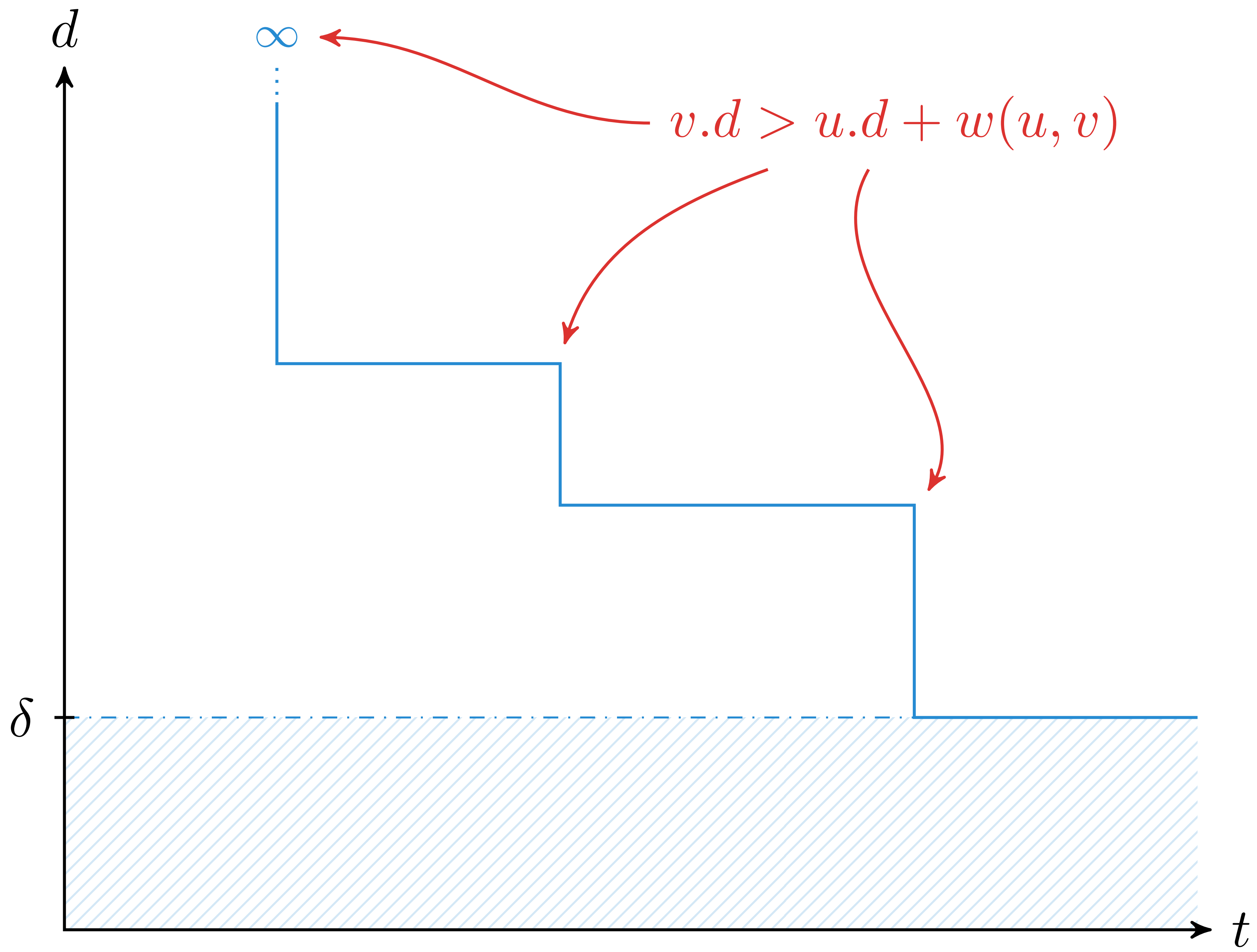
Vi leter etter bedre veier; $v.d$ er best så langt



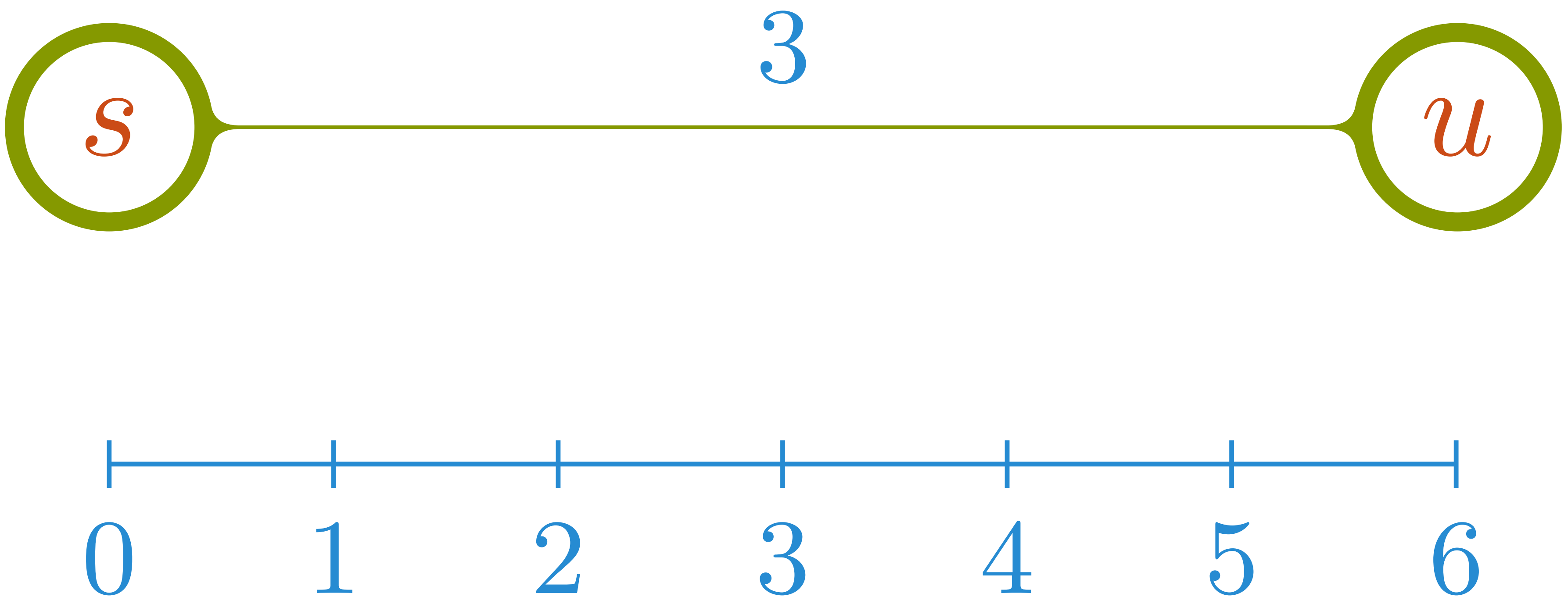
Vi kan naturligvis aldri få $v.d$ mindre enn $\delta(s, v)$



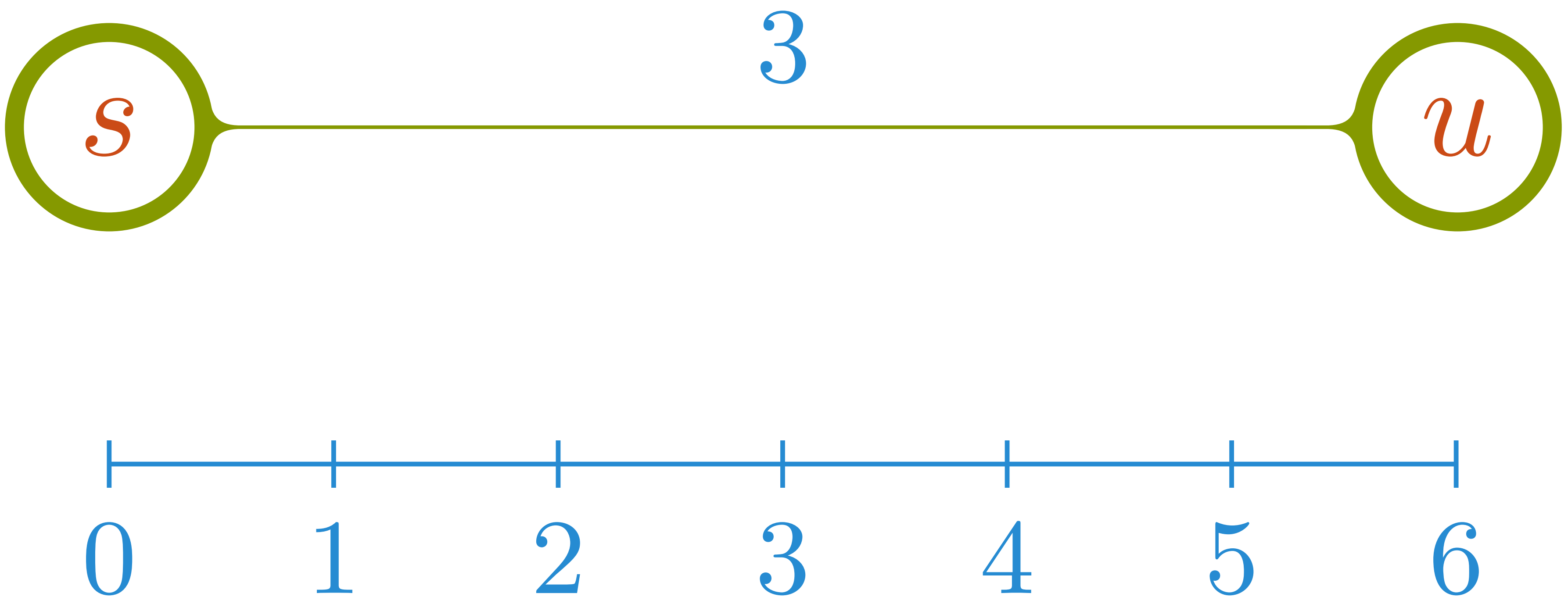
Hver gang vi finner en snarvei $s \rightsquigarrow u \rightarrow v$, synker estimatet



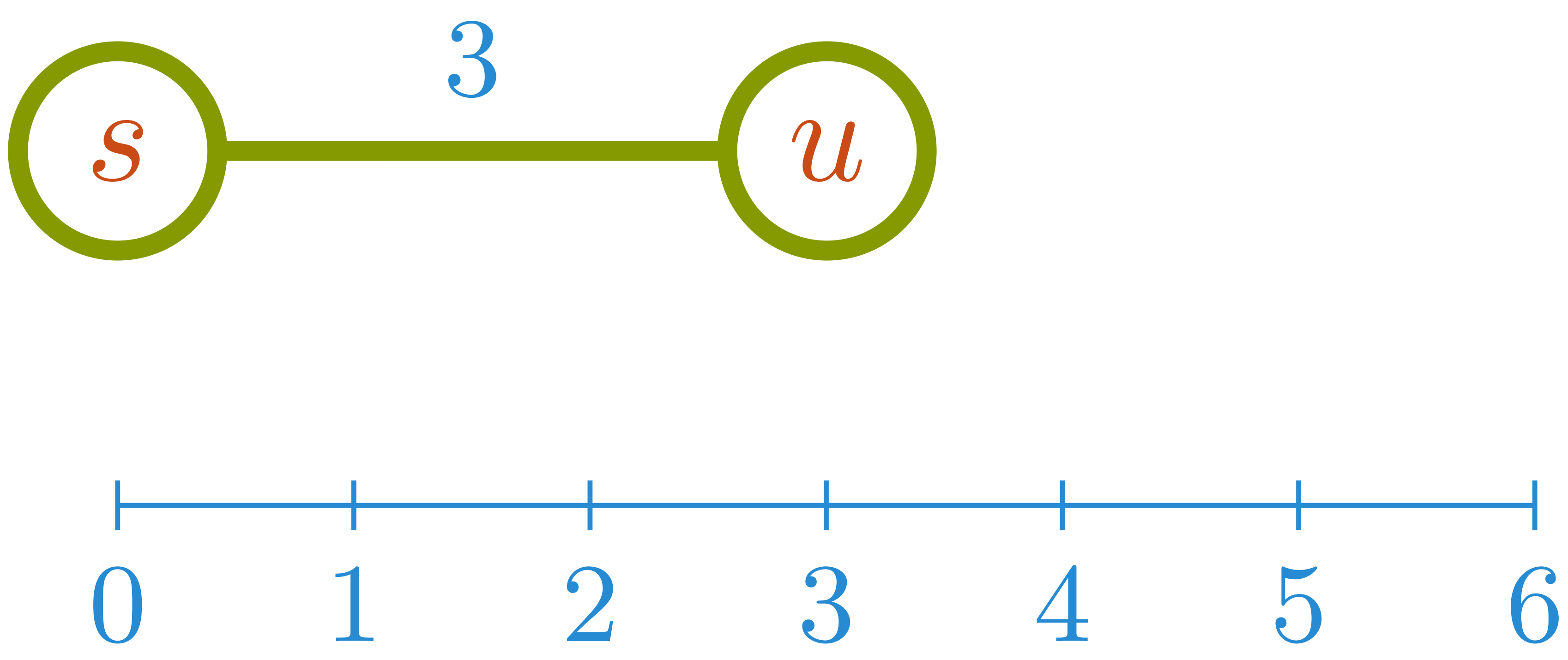
Hver gang vi finner en snarvei $s \rightsquigarrow u \to v$, synker estimatet



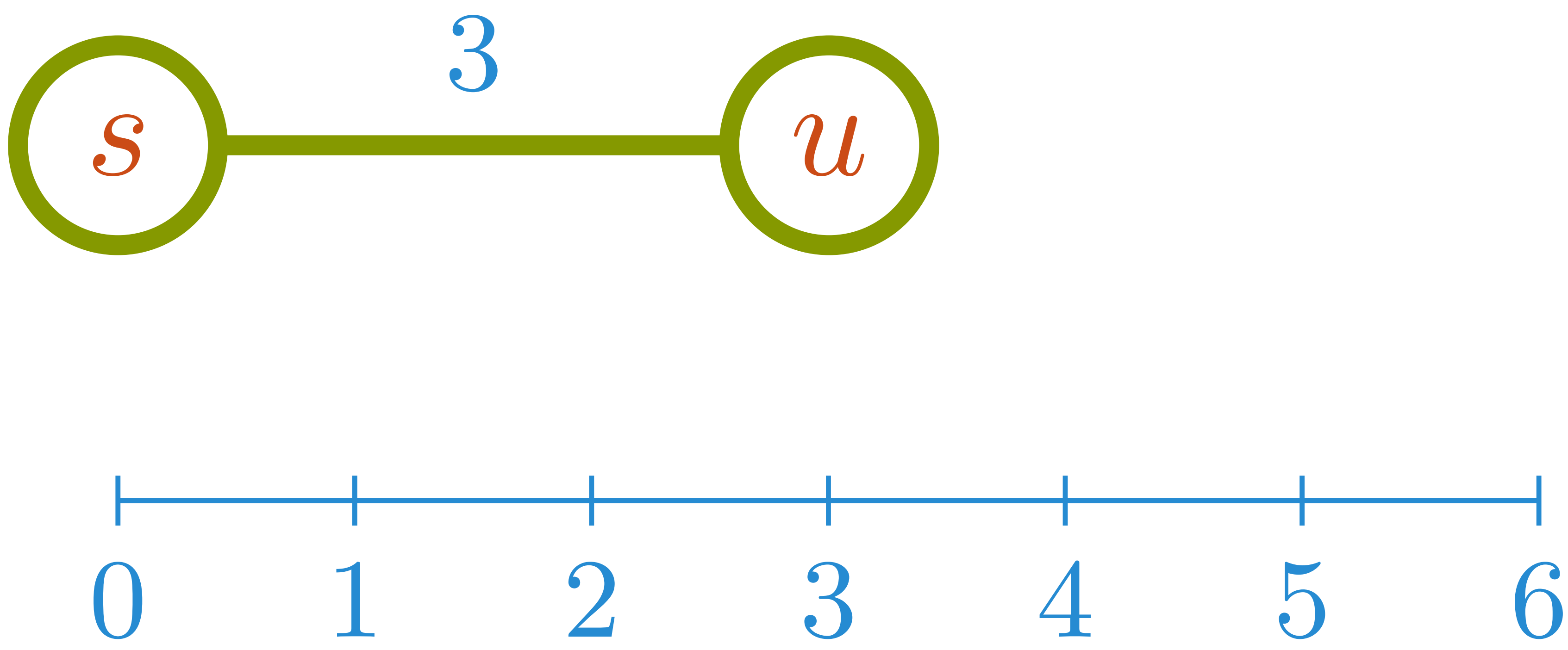
Her er s startnoden, og avstans-overestimatet $u.d$ er 6



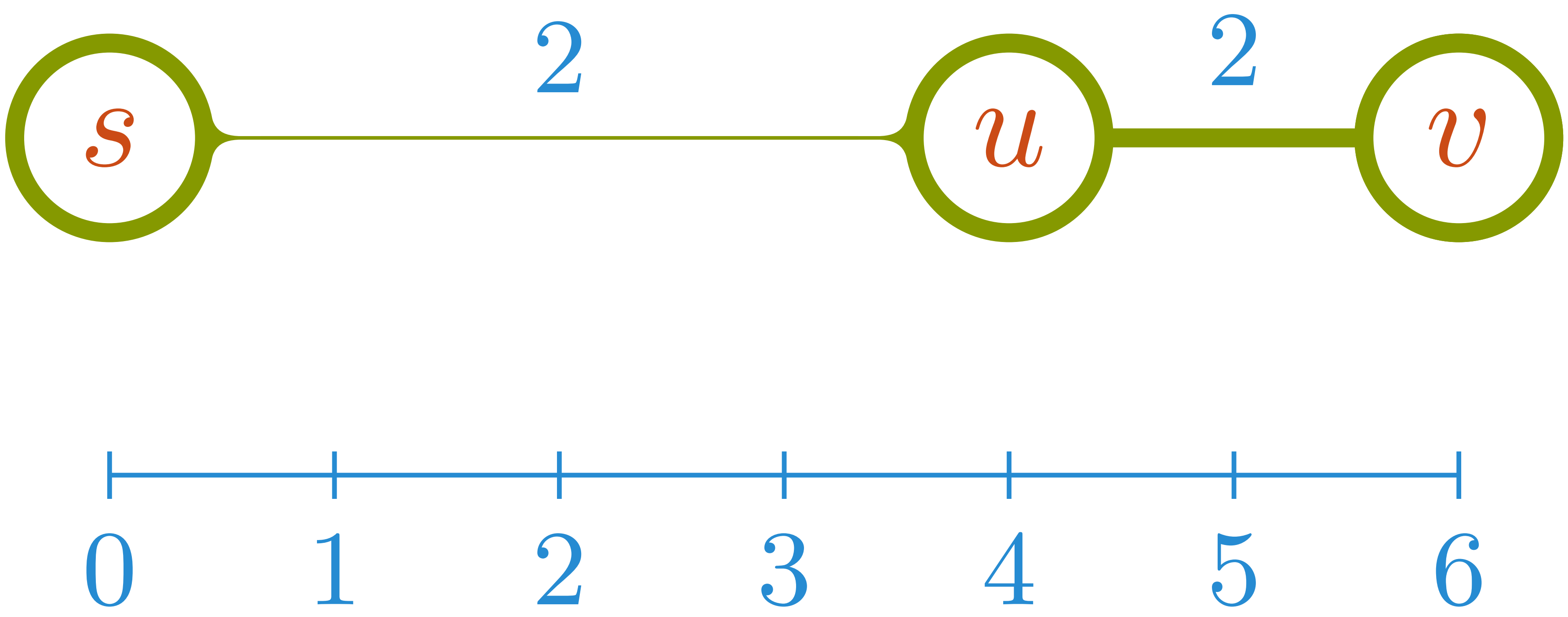
Men esimtatet her trenger ikke være mer enn 3



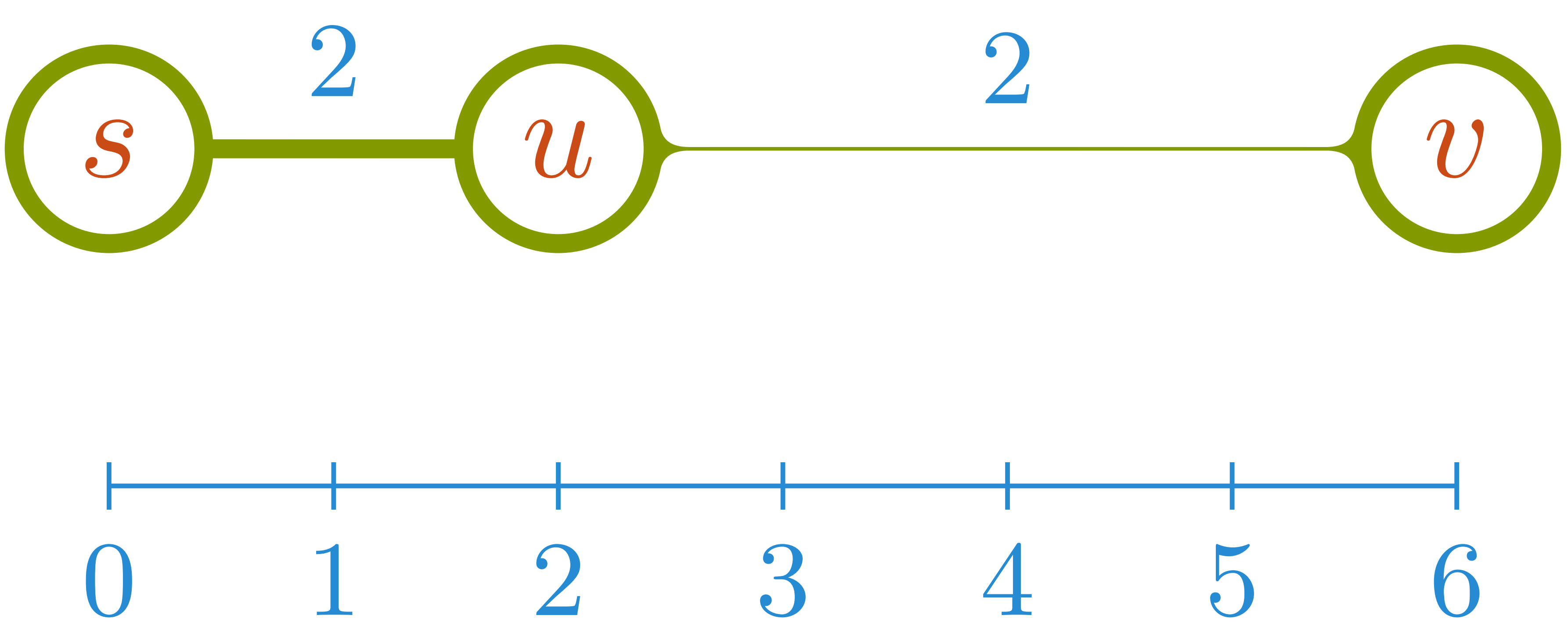
Men esimtatet her trenger ikke være mer enn 3



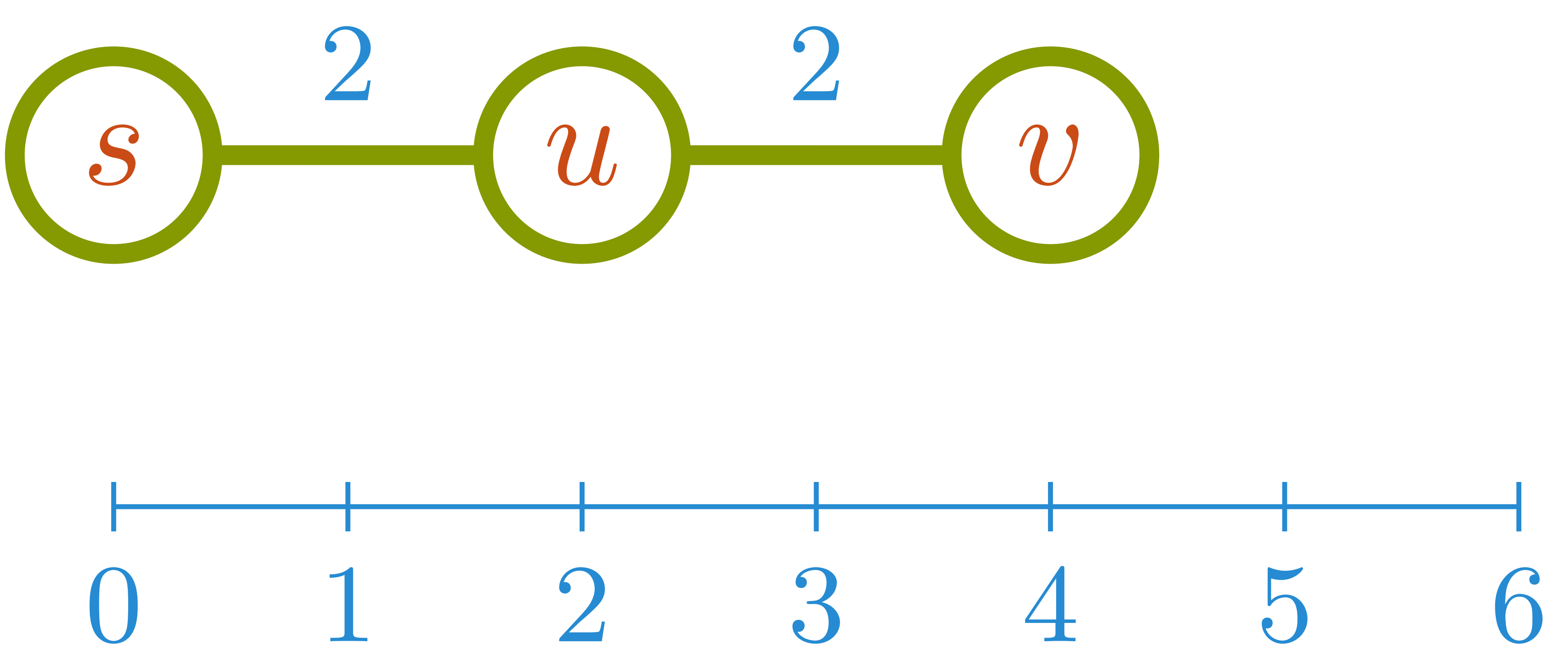
Kanskje det finnes en kortere vei, men avstanden er maks 3



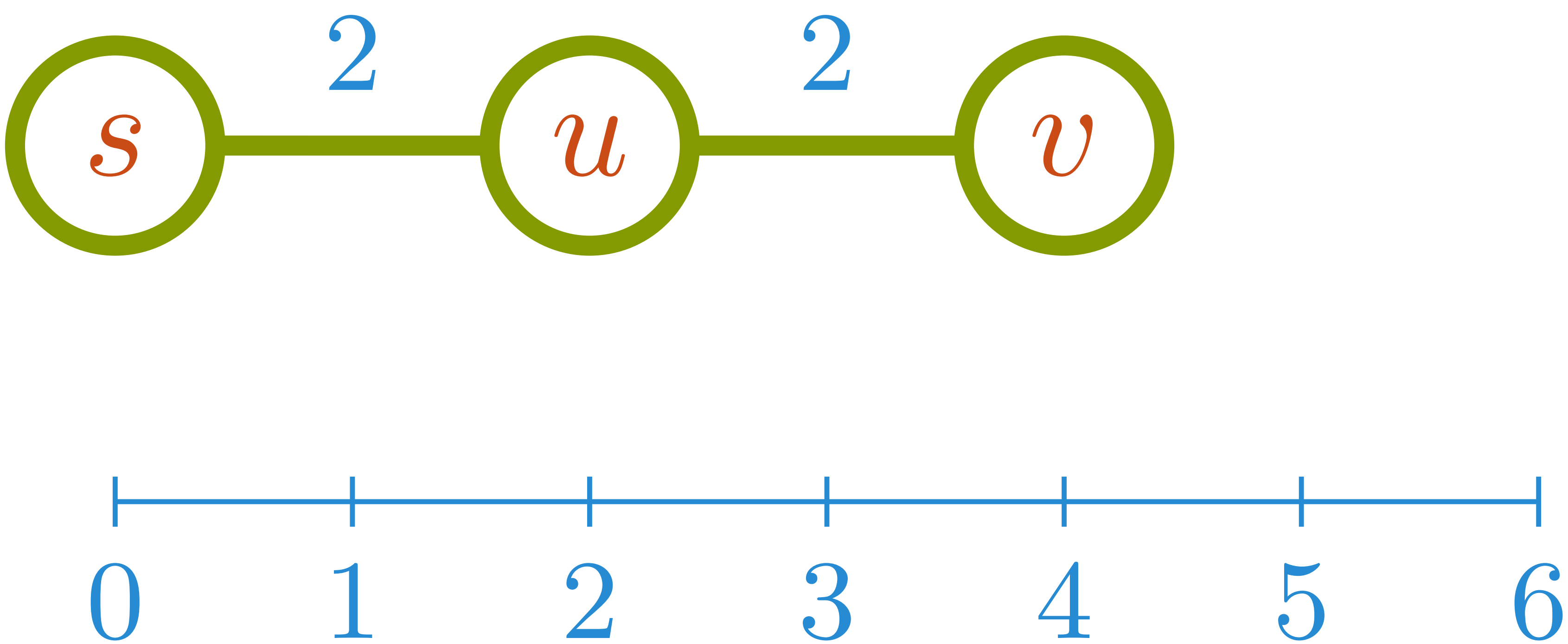
Her er $u.d$ for stor; trenger maks være 2



Men nå ser vi at $v.d$ er for stor; trenger maks være $d.u + 2 = 4$

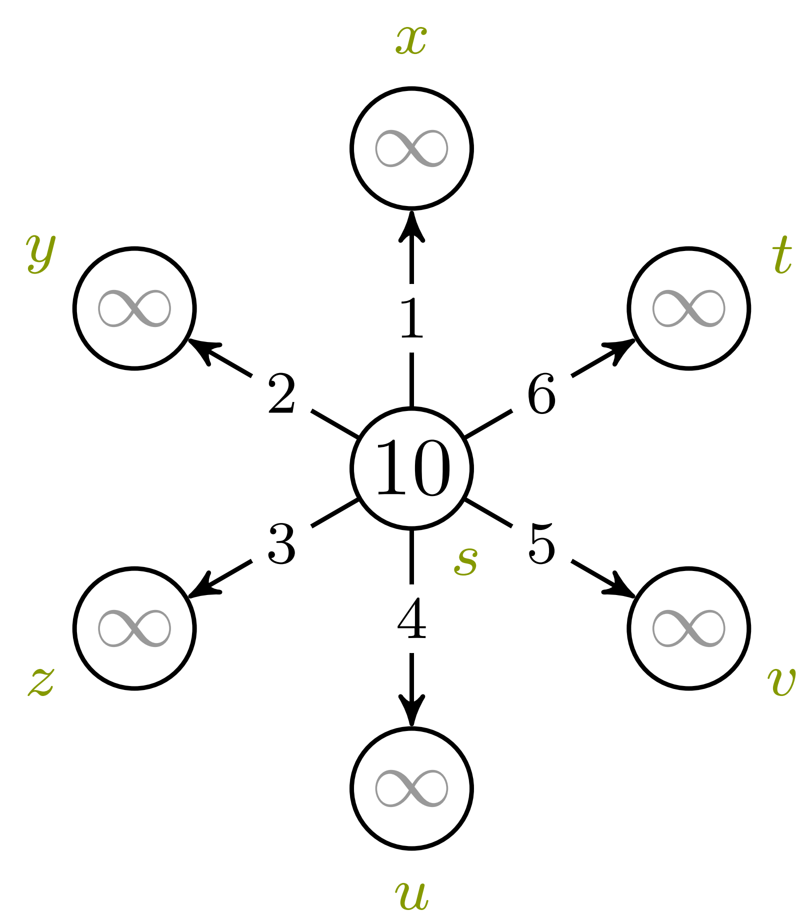


Finnes andre kanter, så kan veien være kortere; ikke lengre!

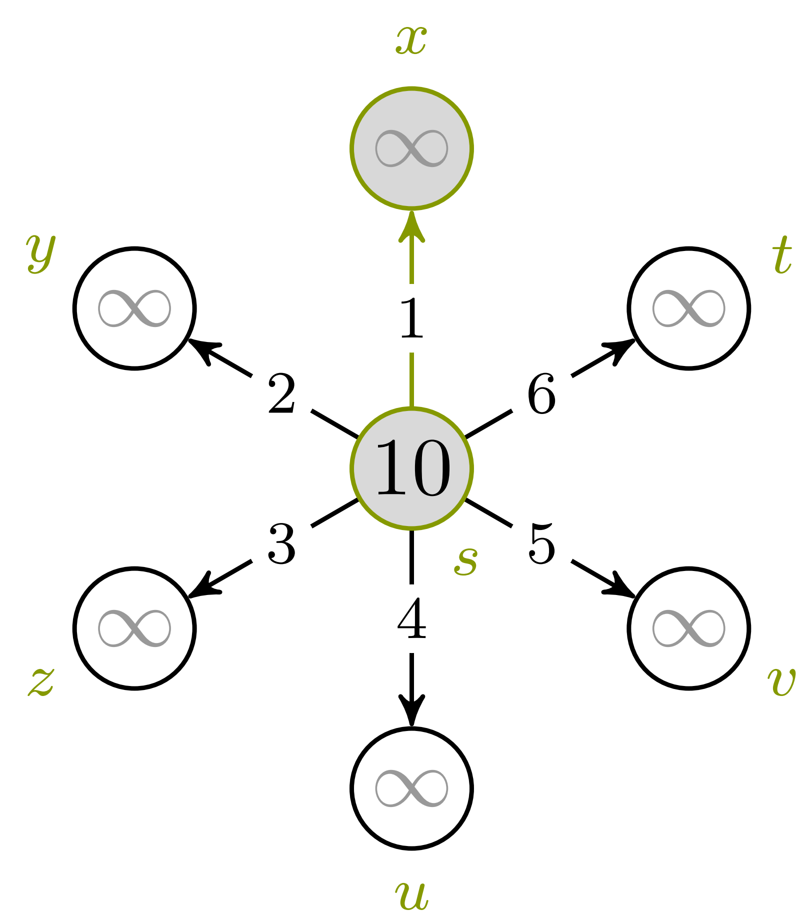


Er dette den korteste stien, så er $u.d$ og $v.d$ nå korrekte

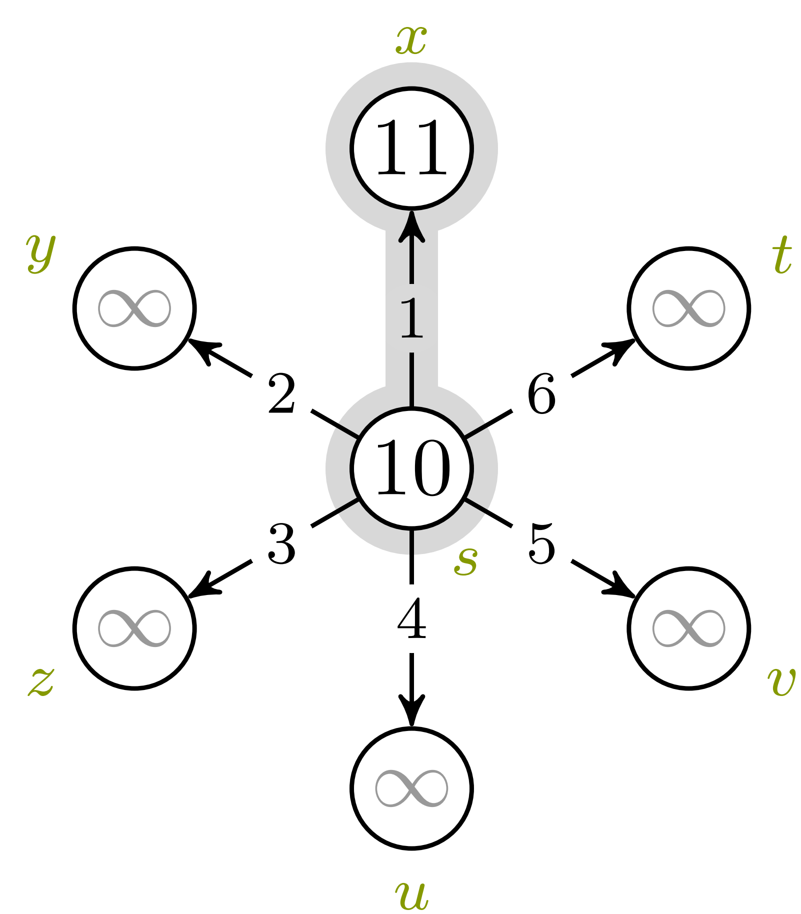
RELAX(s, x)
RELAX(s, y)
RELAX(s, z)
RELAX(s, u)
RELAX(s, v)
RELAX(s, t)



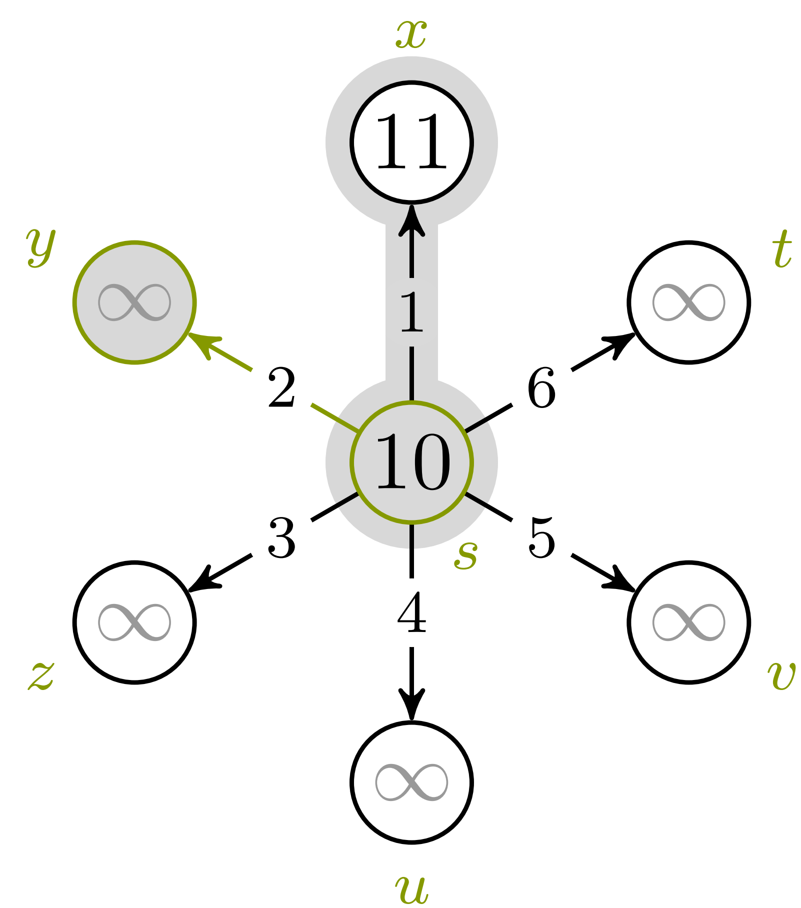
1 Relax(s, x)
2 Relax(s, y)
3 Relax(s, z)
4 Relax(s, u)
5 Relax(s, v)
6 Relax(s, t)



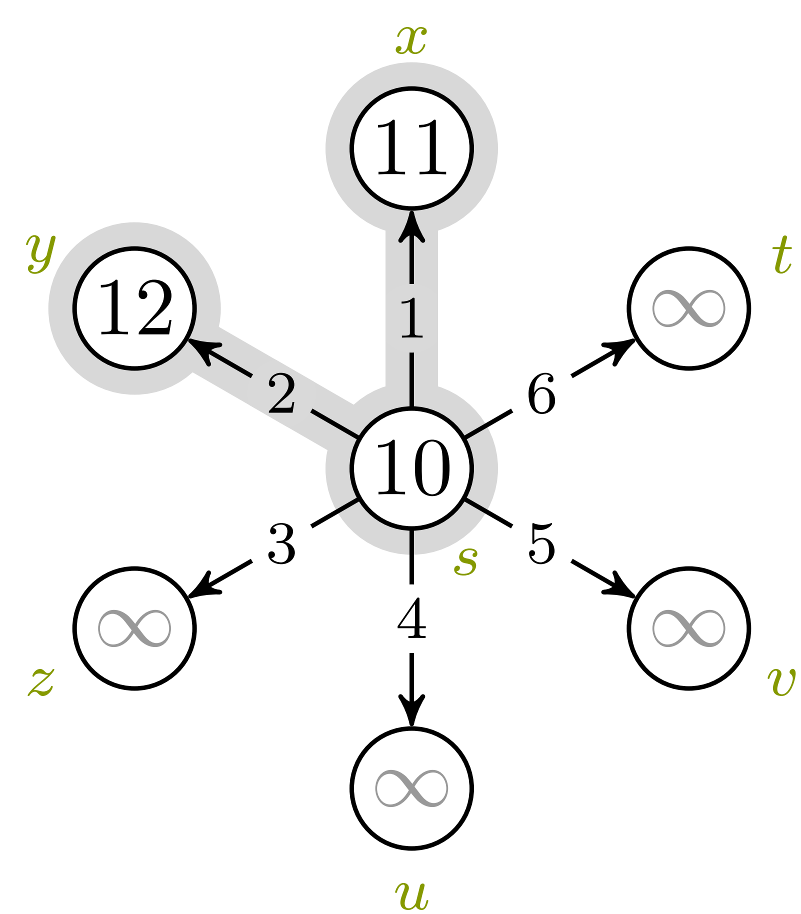
1 RELAX(s, x)
2 RELAX(s, y)
3 RELAX(s, z)
4 RELAX(s, u)
5 RELAX(s, v)
6 RELAX(s, t)



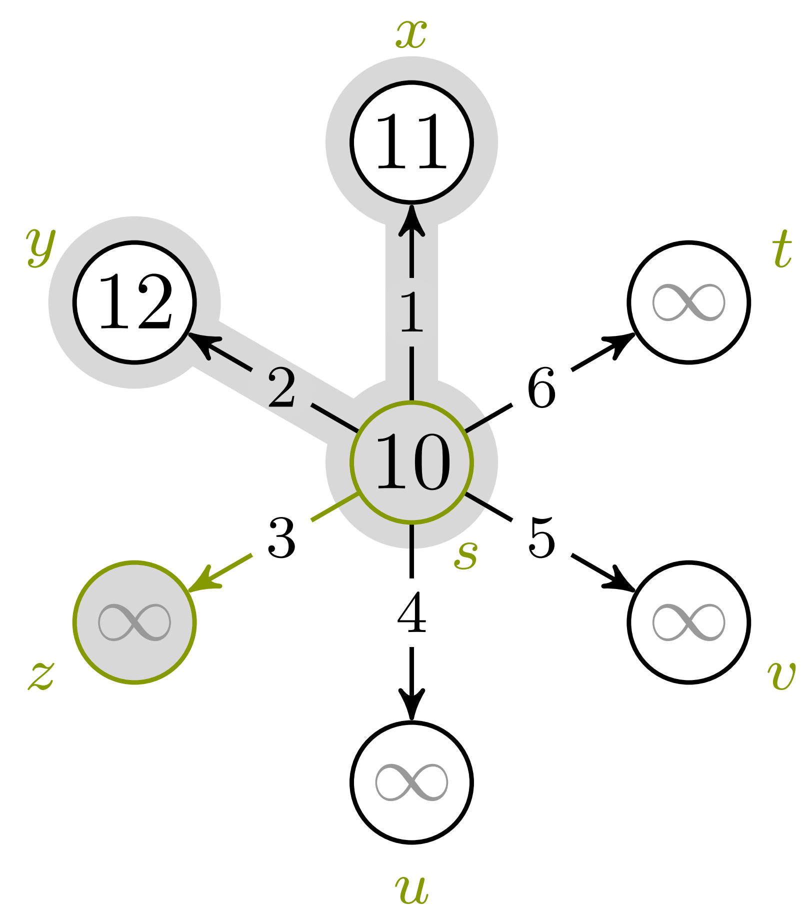
1 RELAX(s, x)
2 RELAX(s, y)
3 RELAX(s, z)
4 RELAX(s, u)
5 RELAX(s, v)
6 RELAX(s, t)



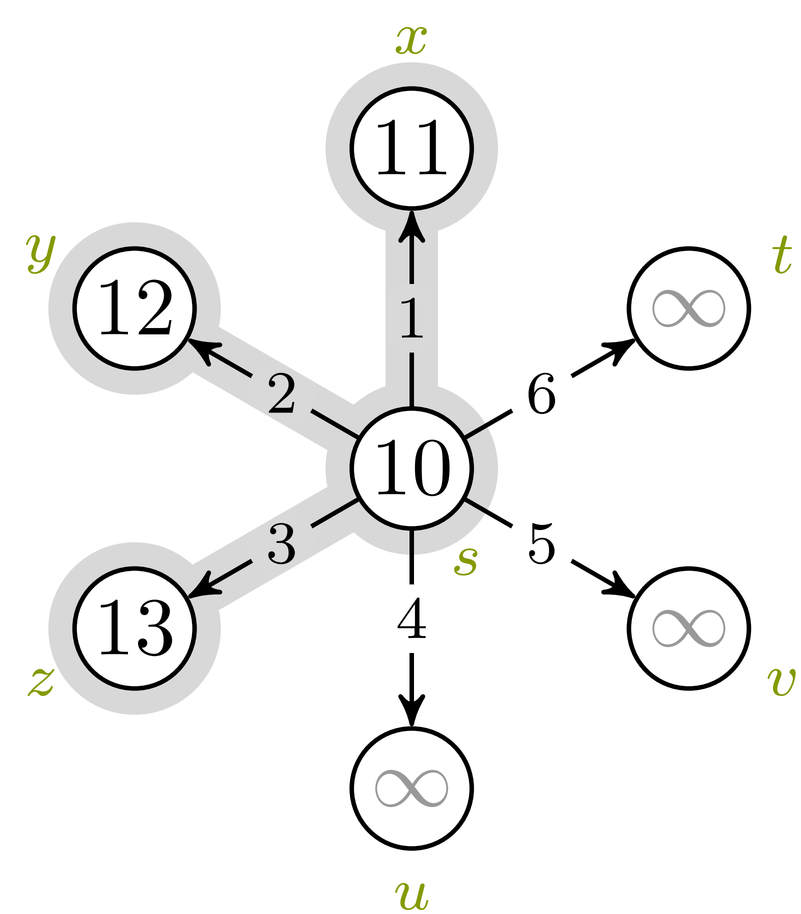
```
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)
```



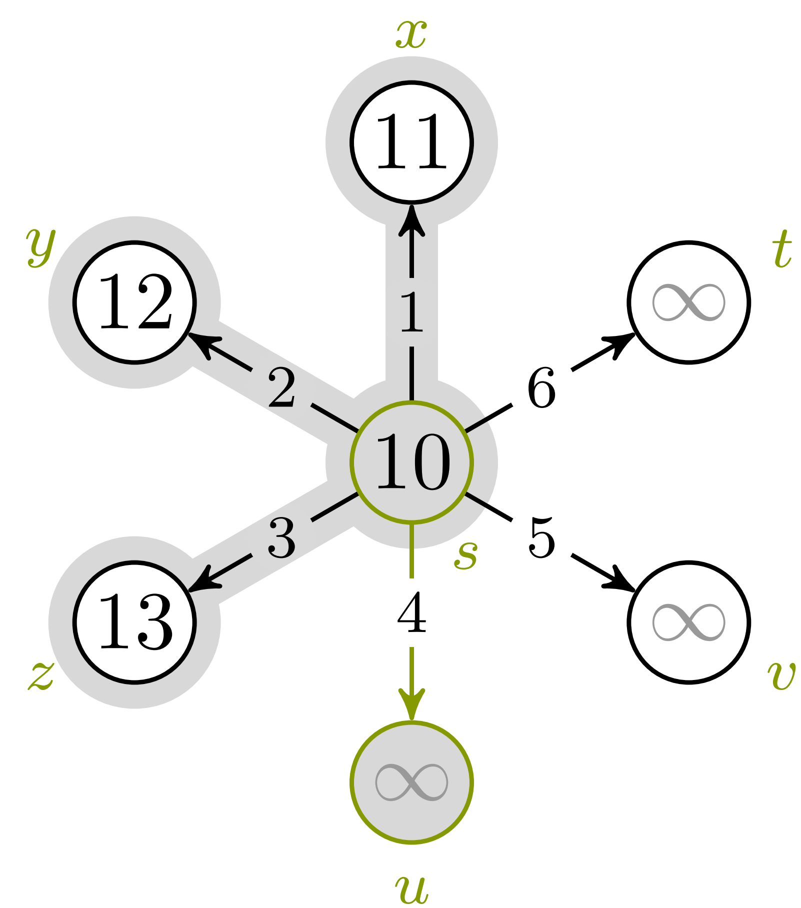
```
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)
```



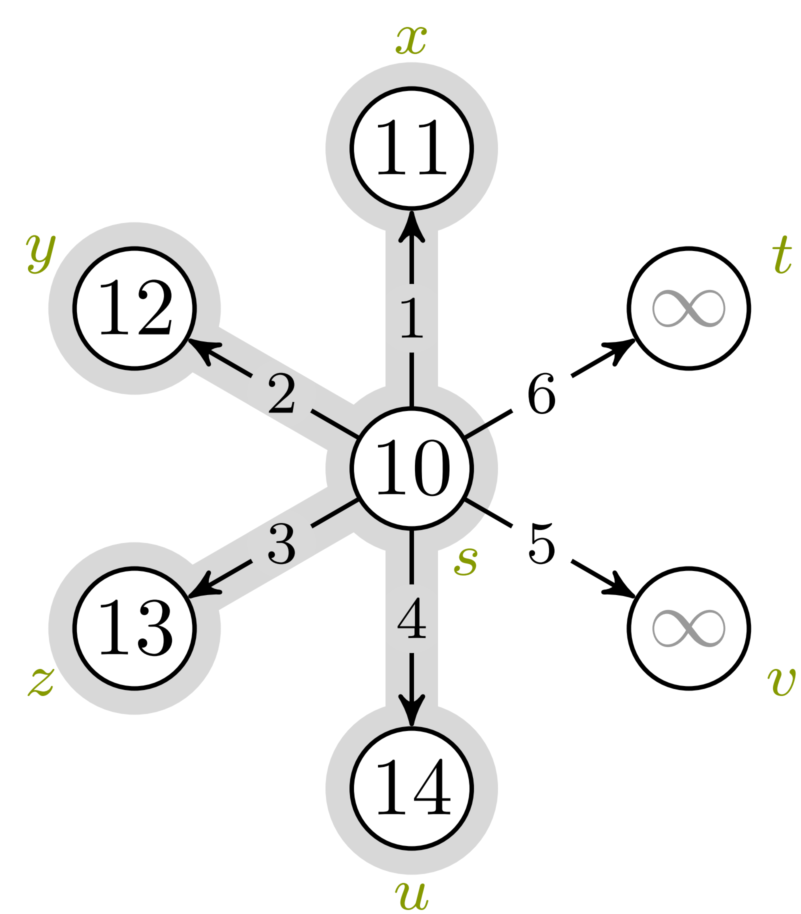
```
RELAX(s, x)
RELAX(s, y)
RELAX(s, z)
RELAX(s, u)
RELAX(s, v)
RELAX(s, t)
```



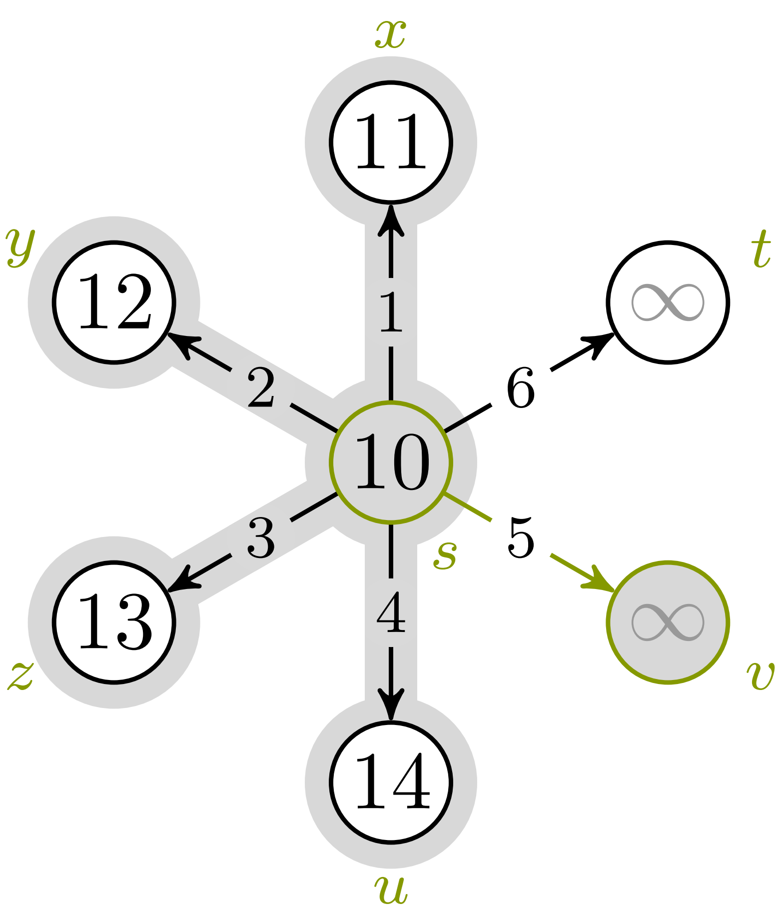
$\text{Relax}(s, x)$
$\text{Relax}(s, y)$
$\text{Relax}(s, z)$
$\text{Relax}(s, u)$
$\text{Relax}(s, v)$
$\text{Relax}(s, t)$



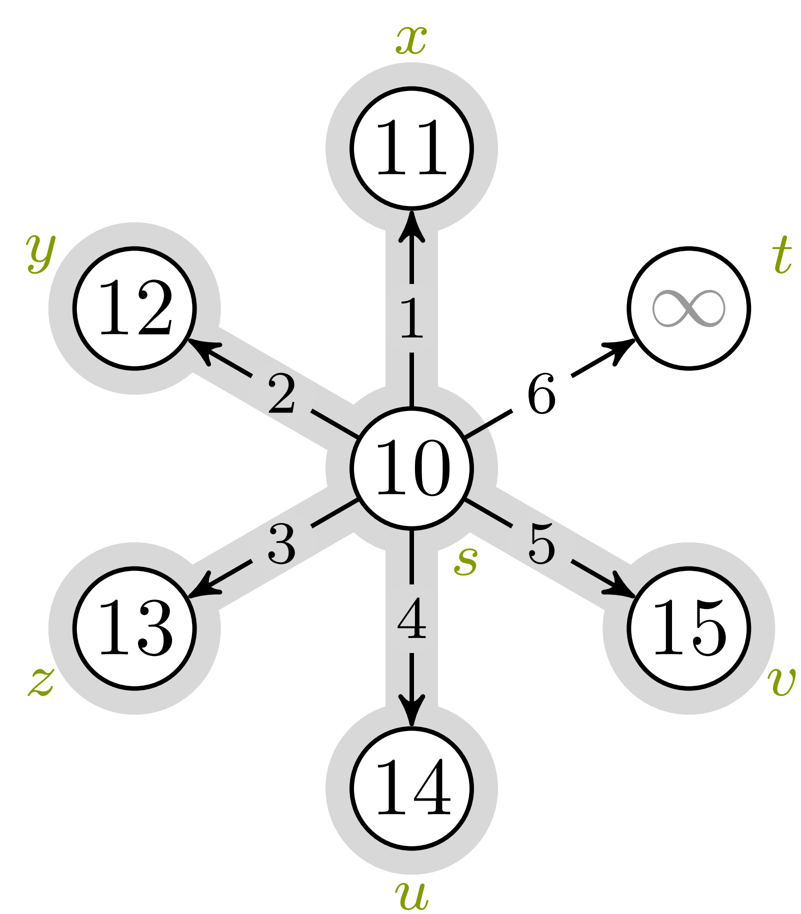
```
RELAX(s, x)
RELAX(s, y)
RELAX(s, z)
RELAX(s, u)
RELAX(s, v)
RELAX(s, t)
```



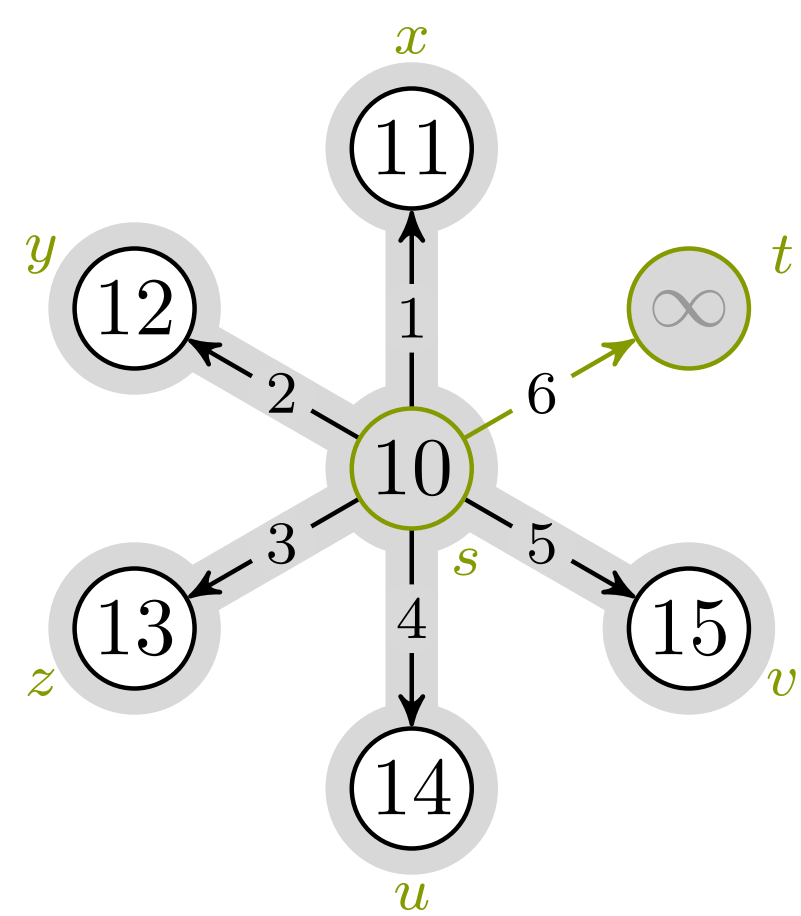
1 Relax(s, x)
2 Relax(s, y)
3 Relax(s, z)
4 Relax(s, u)
5 Relax(s, v)
6 Relax(s, t)



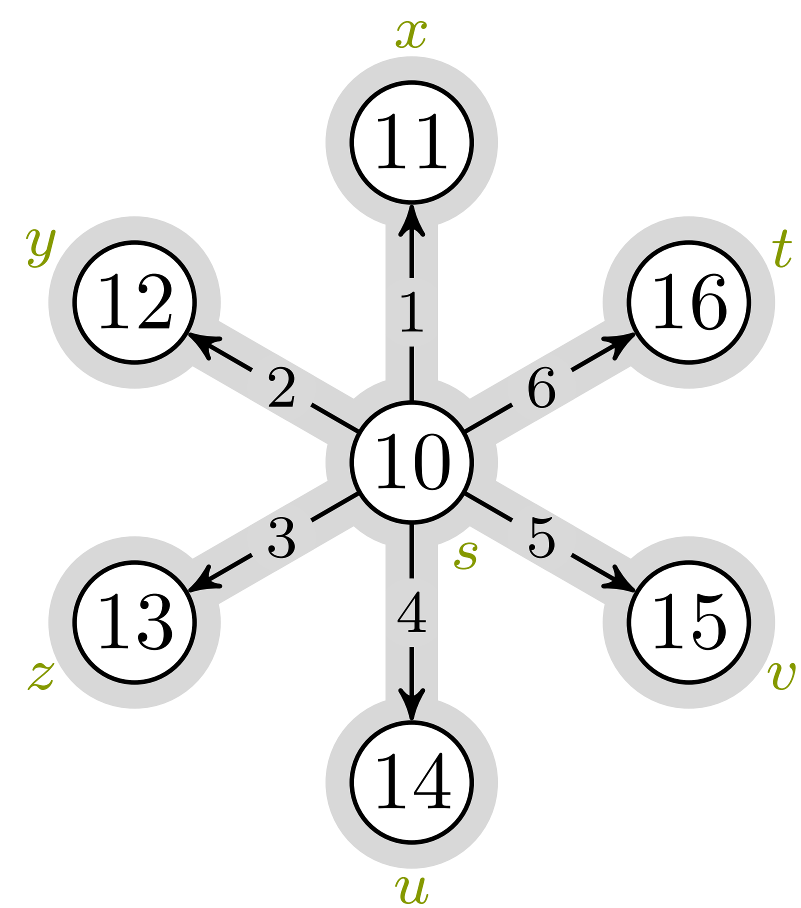
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)



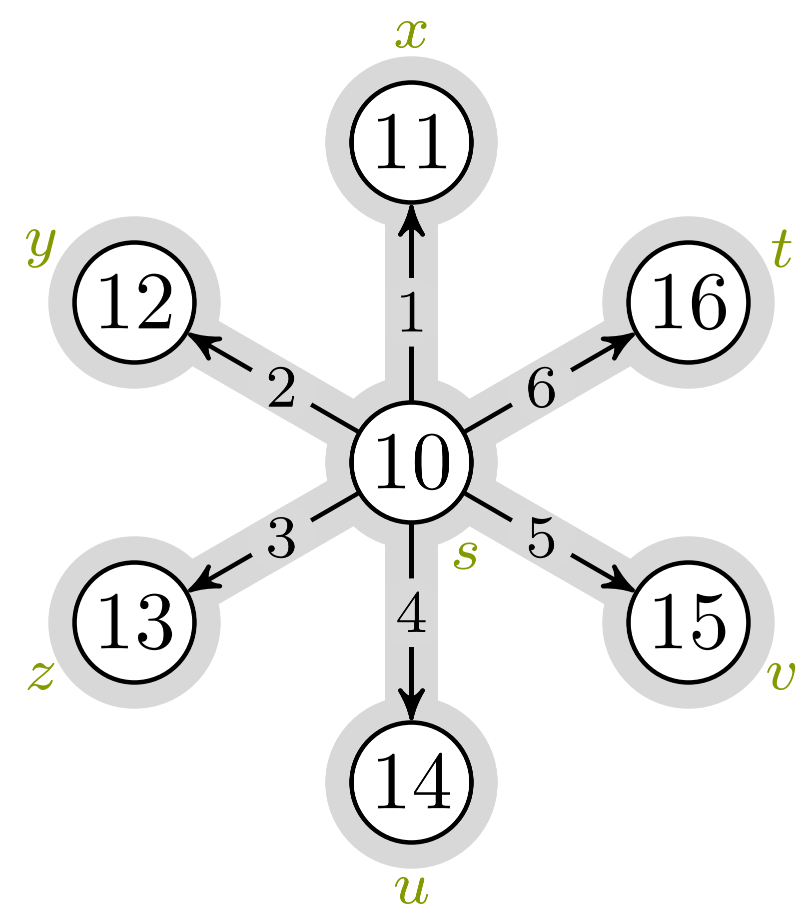
$\text{Relax}(s, x)$
$\text{Relax}(s, y)$
$\text{Relax}(s, z)$
$\text{Relax}(s, u)$
$\text{Relax}(s, v)$
$\text{Relax}(s, t)$



1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(s, y)$
3 $\textsc{Relax}(s, z)$
4 $\textsc{Relax}(s, u)$
5 $\textsc{Relax}(s, v)$
6 $\textsc{Relax}(s, t)$

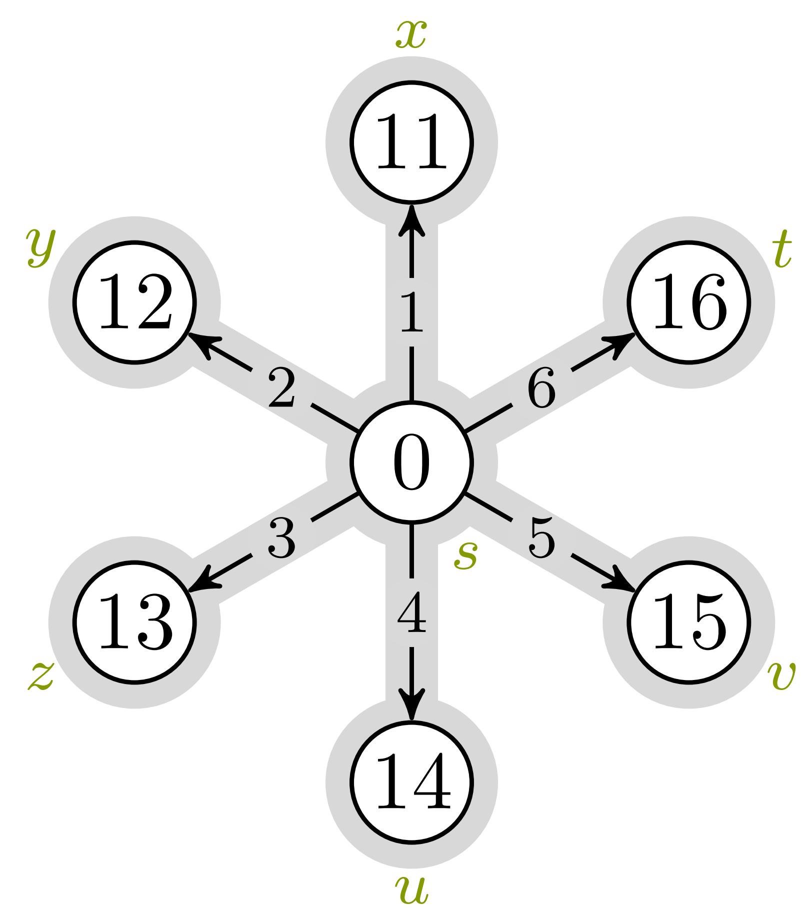


1 Relax(s, x)
2 Relax(s, y)
3 Relax(s, z)
4 Relax(s, u)
5 Relax(s, v)
6 Relax(s, t)



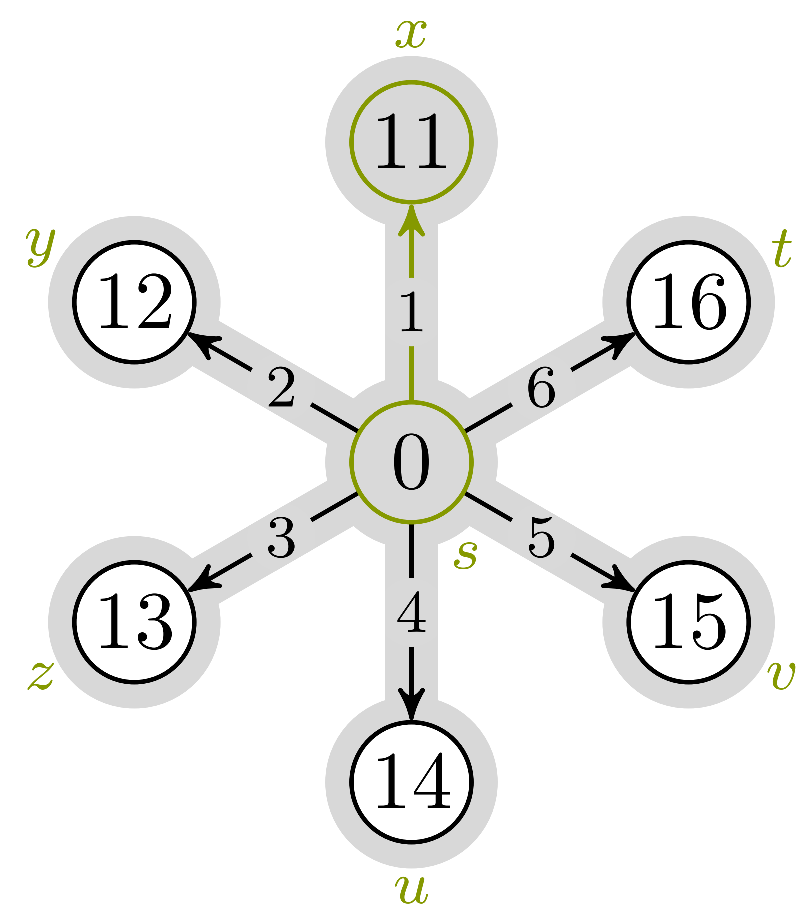
Ferdig . . .

1 Relax(s, x)
2 Relax(s, y)
3 Relax(s, z)
4 Relax(s, u)
5 Relax(s, v)
6 Relax(s, t)

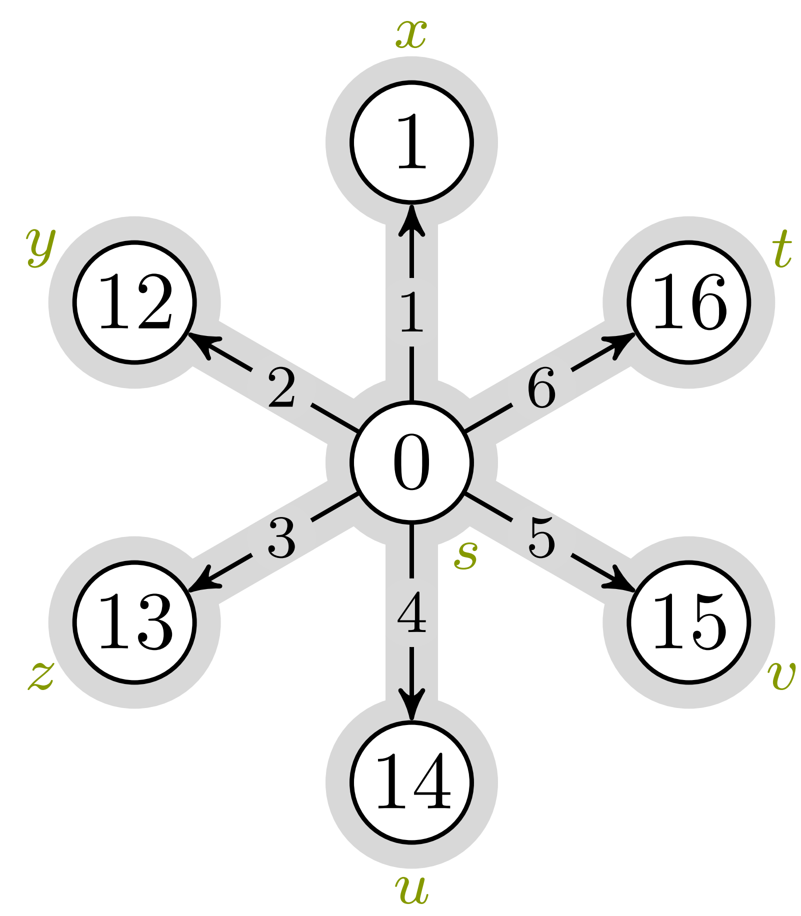


…med mindre $s.d$ endres!

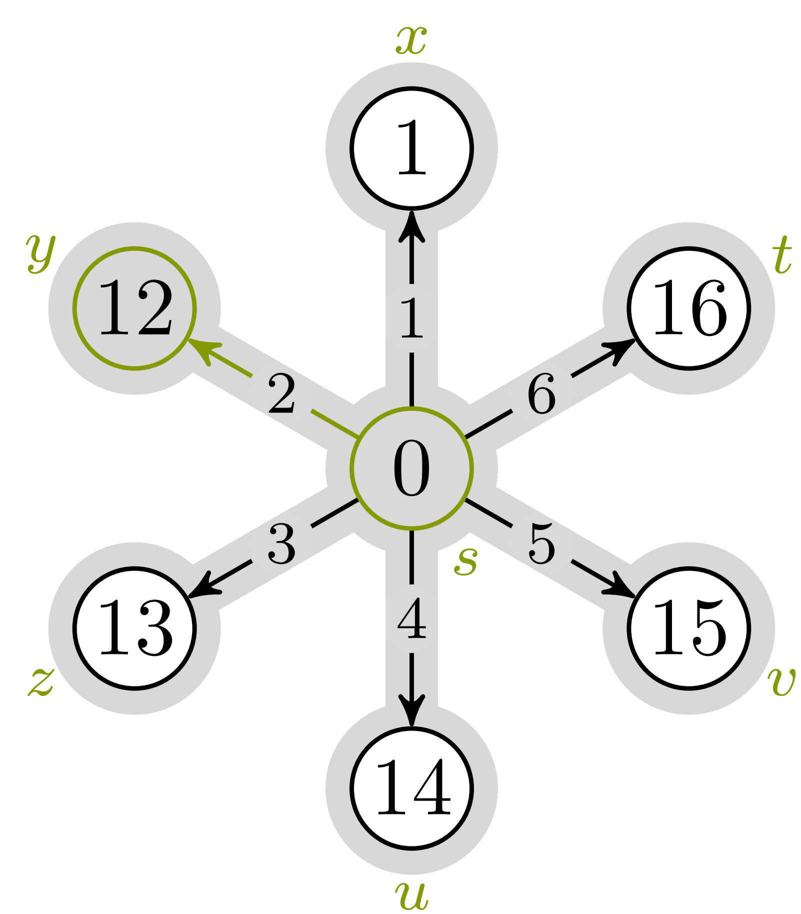
1 RELAX(s, x)
2 RELAX(s, y)
3 RELAX(s, z)
4 RELAX(s, u)
5 RELAX(s, v)
6 RELAX(s, t)



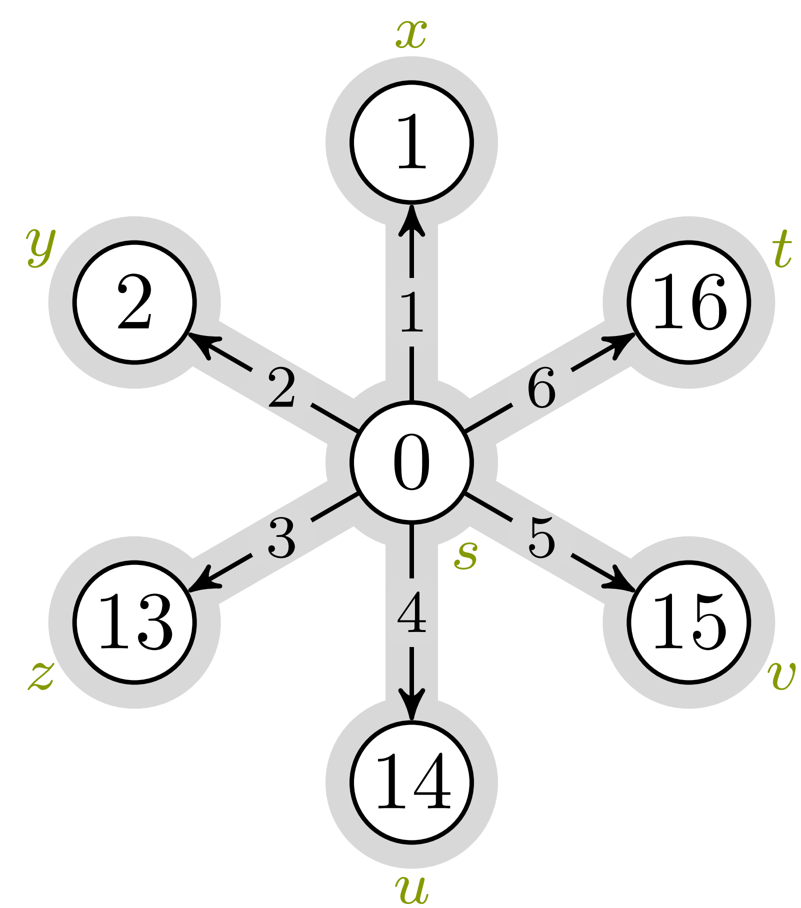
1 RELAX(s, x)
2 RELAX(s, y)
3 RELAX(s, z)
4 RELAX(s, u)
5 RELAX(s, v)
6 RELAX(s, t)



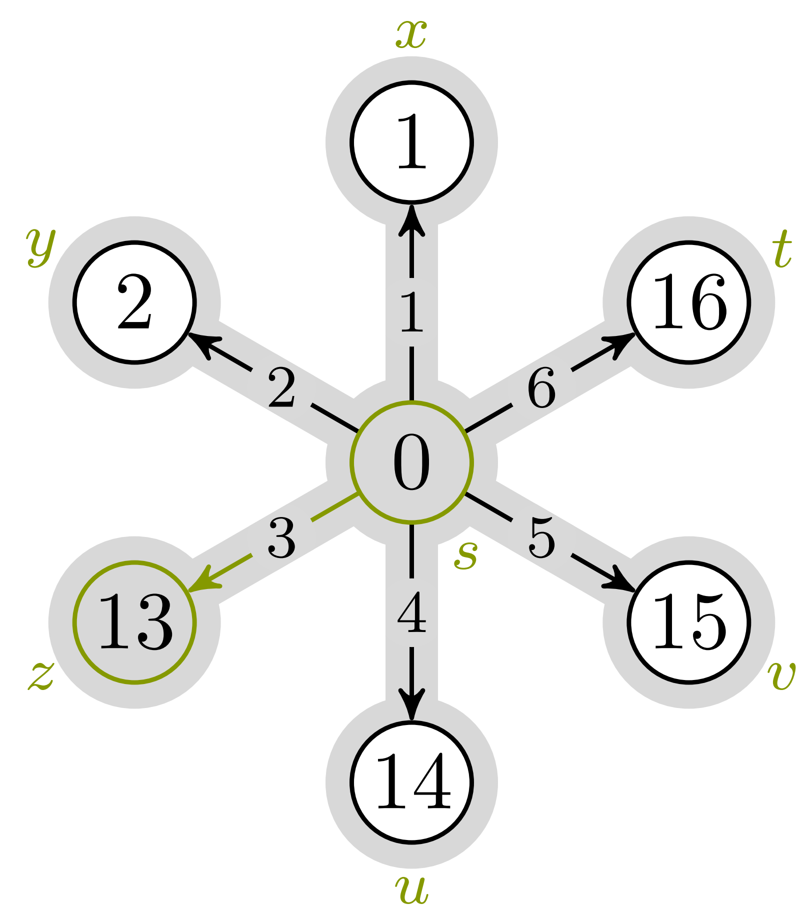
```
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)
```



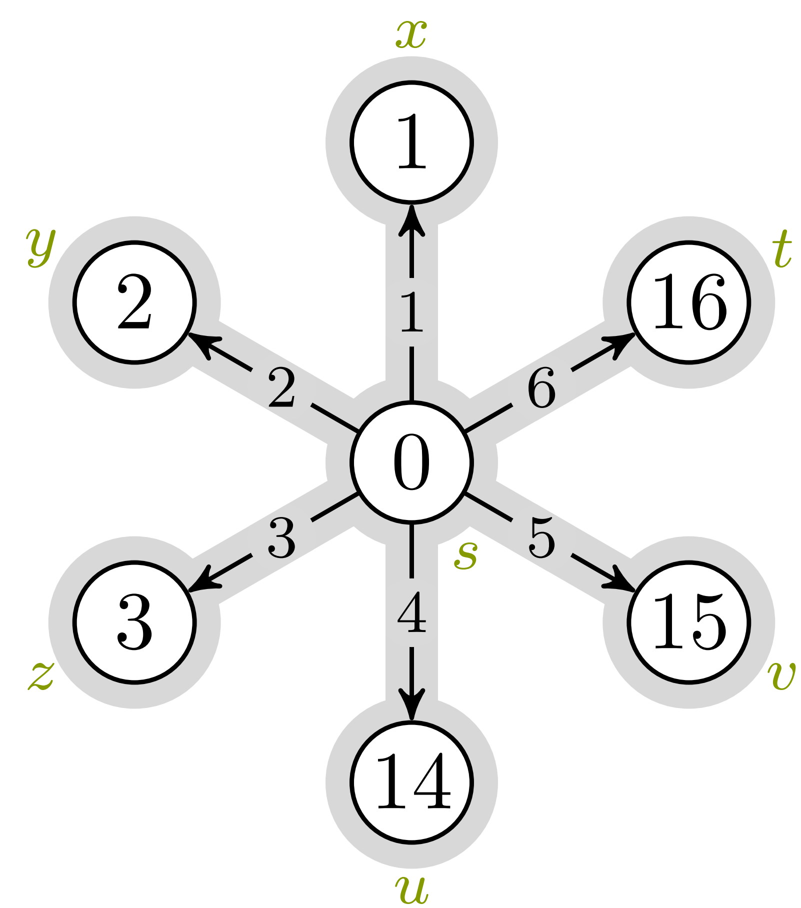
```
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)
```



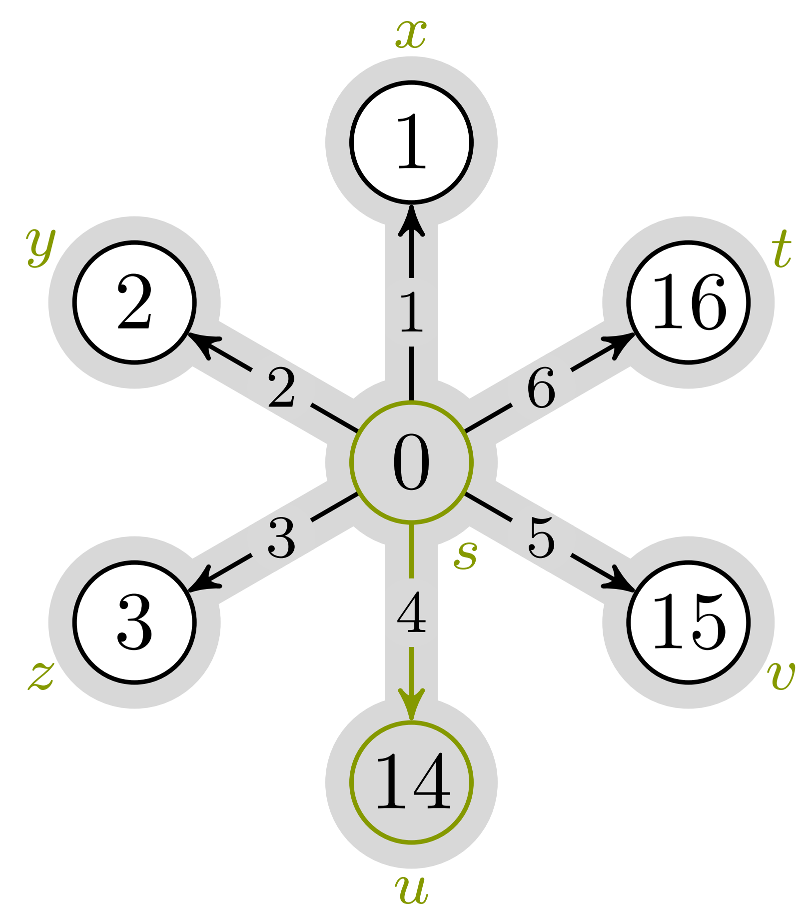
$\textsc{Relax}(s, x)$
$\textsc{Relax}(s, y)$
$\textsc{Relax}(s, z)$
$\textsc{Relax}(s, u)$
$\textsc{Relax}(s, v)$
$\textsc{Relax}(s, t)$



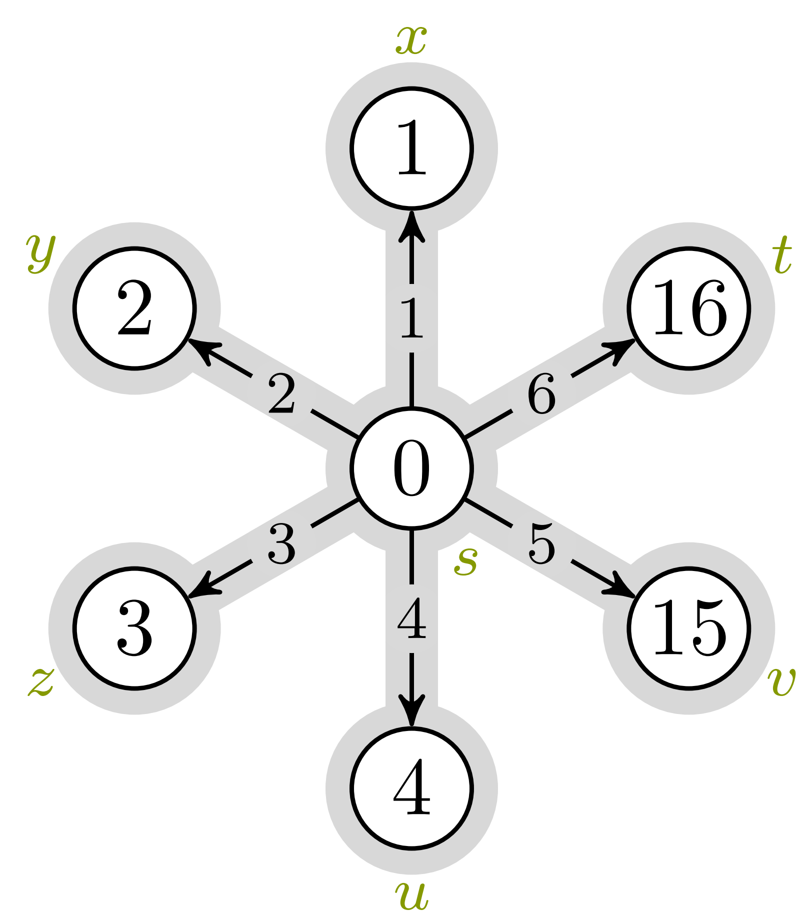
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)



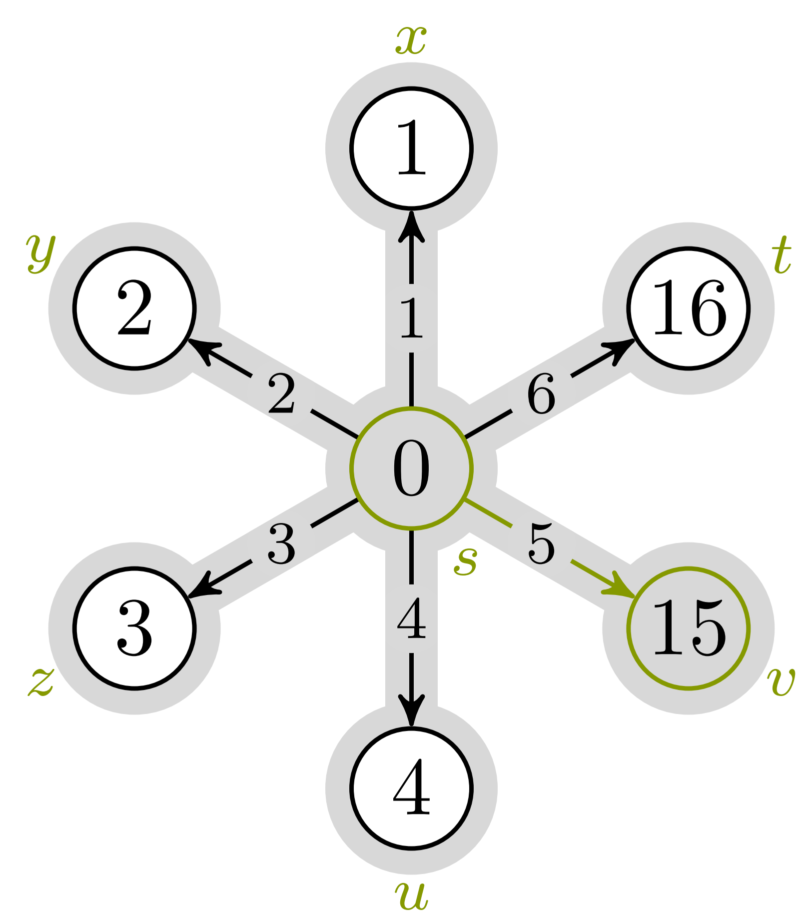
1 RELAX(s, x)
2 RELAX(s, y)
3 RELAX(s, z)
4 RELAX(s, u)
5 RELAX(s, v)
6 RELAX(s, t)



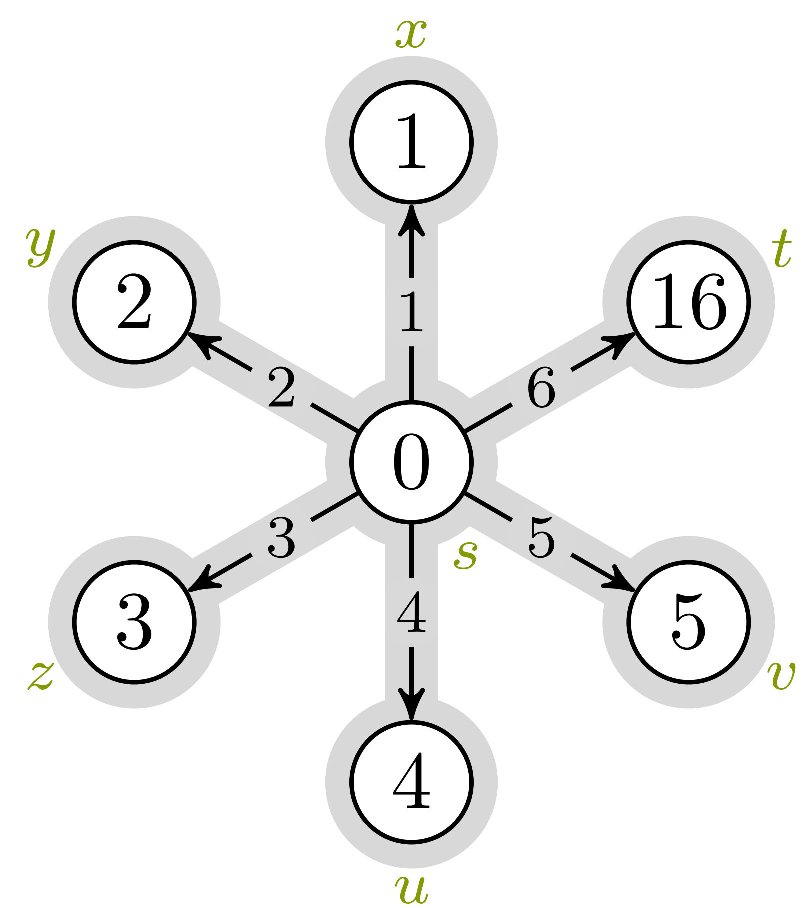
$\textsc{Relax}(s, x)$
$\textsc{Relax}(s, y)$
$\textsc{Relax}(s, z)$
$\textsc{Relax}(s, u)$
$\textsc{Relax}(s, v)$
$\textsc{Relax}(s, t)$



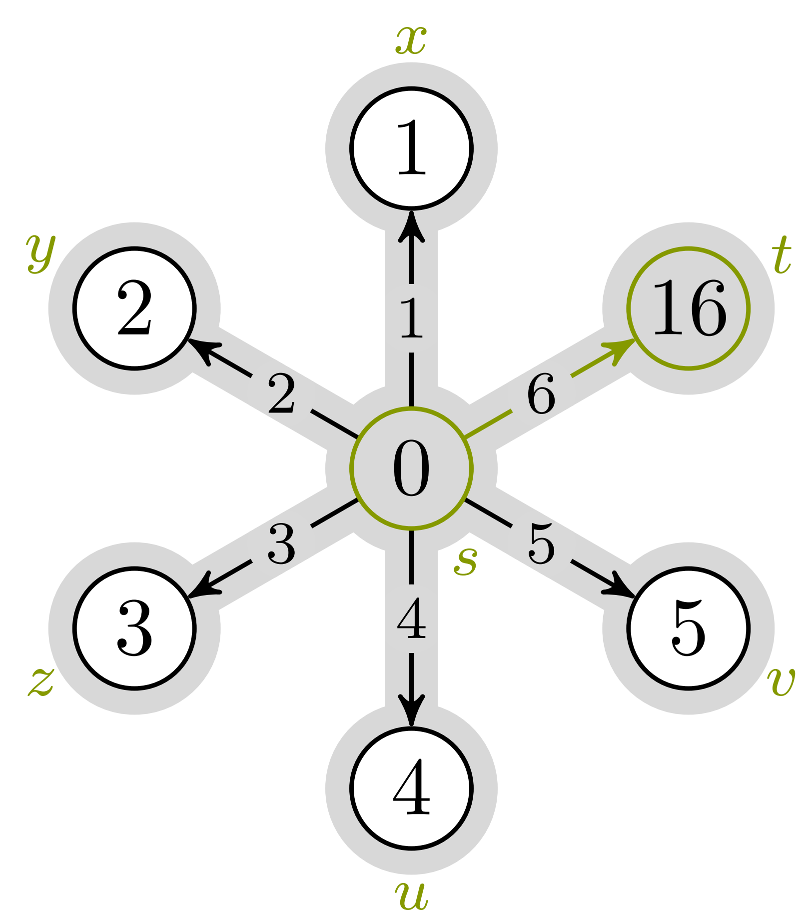
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)



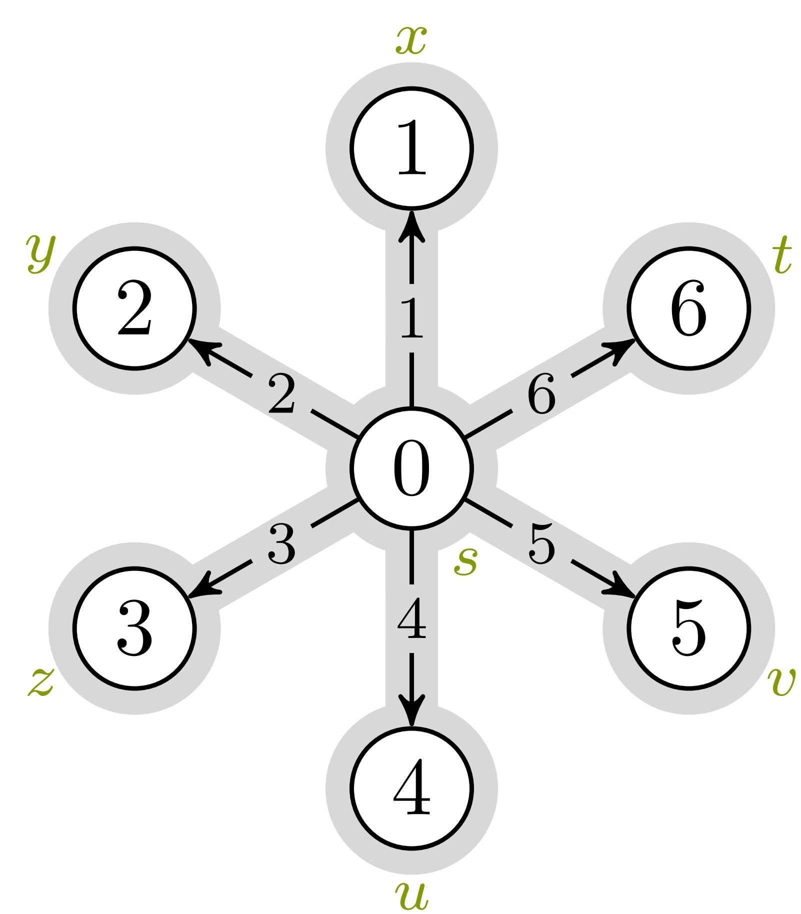
1 RELAX(s, x)
2 RELAX(s, y)
3 RELAX(s, z)
4 RELAX(s, u)
5 RELAX(s, v)
6 RELAX(s, t)



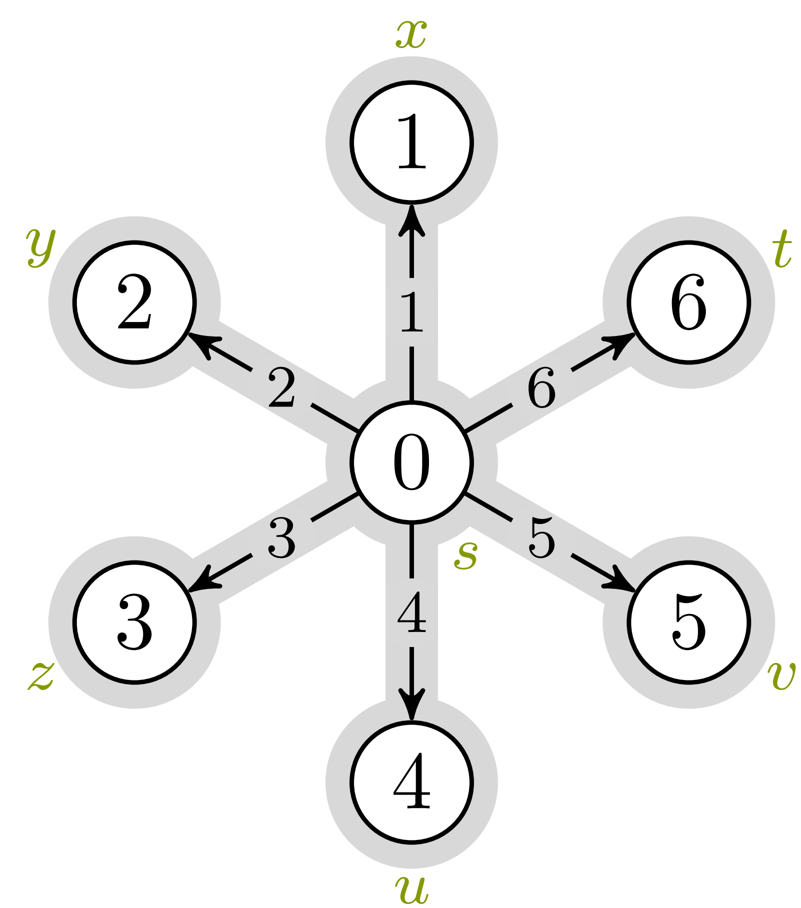
1 RELAX(s, x)
2 RELAX(s, y)
3 RELAX(s, z)
4 RELAX(s, u)
5 RELAX(s, v)
6 RELAX(s, t)



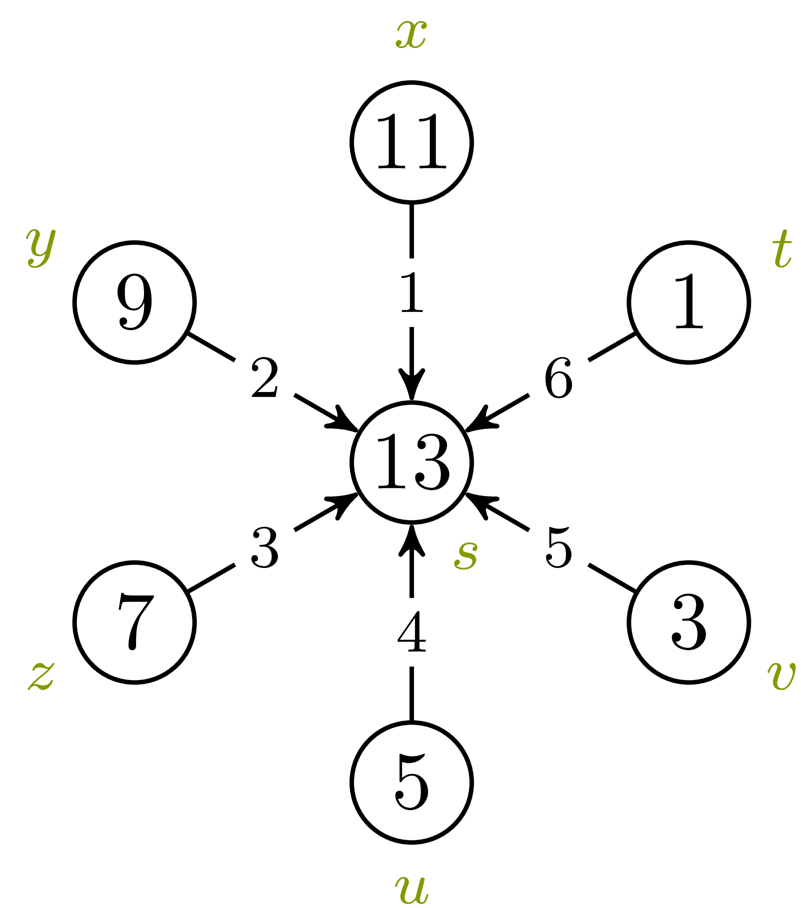
```
Relax(s, x)
Relax(s, y)
Relax(s, z)
Relax(s, u)
Relax(s, v)
Relax(s, t)
```



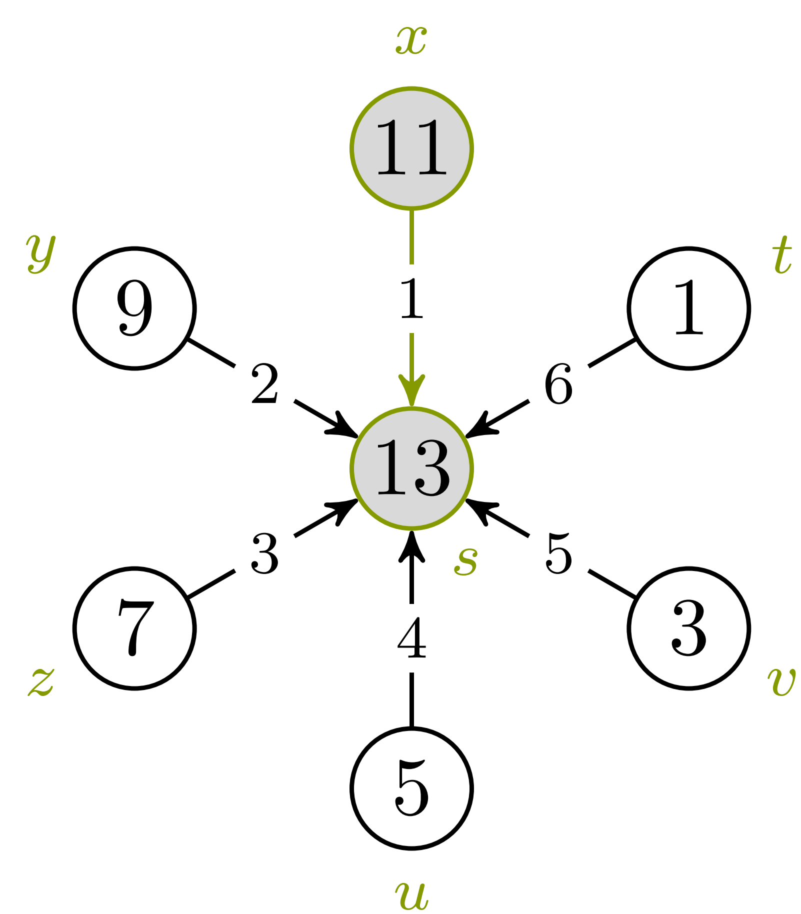
$\textsc{Relax}(s, x)$
$\textsc{Relax}(s, y)$
$\textsc{Relax}(s, z)$
$\textsc{Relax}(s, u)$
$\textsc{Relax}(s, v)$
$\textsc{Relax}(s, t)$



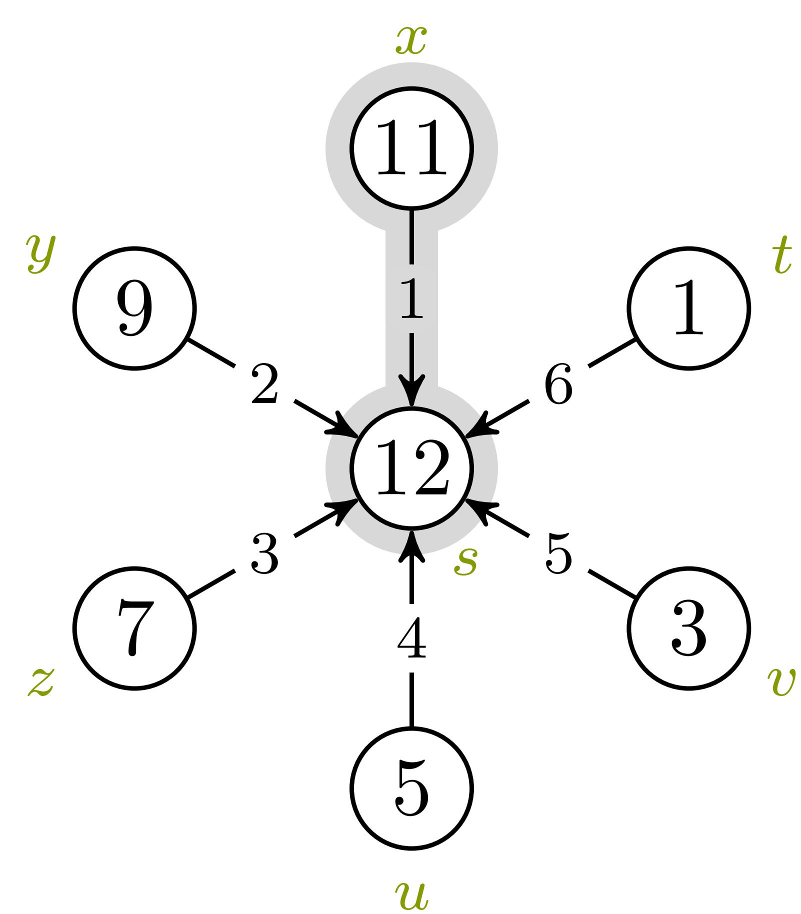
Relax(x, s)
Relax(y, s)
Relax(z, s)
Relax(u, s)
Relax(v, s)
Relax(t, s)



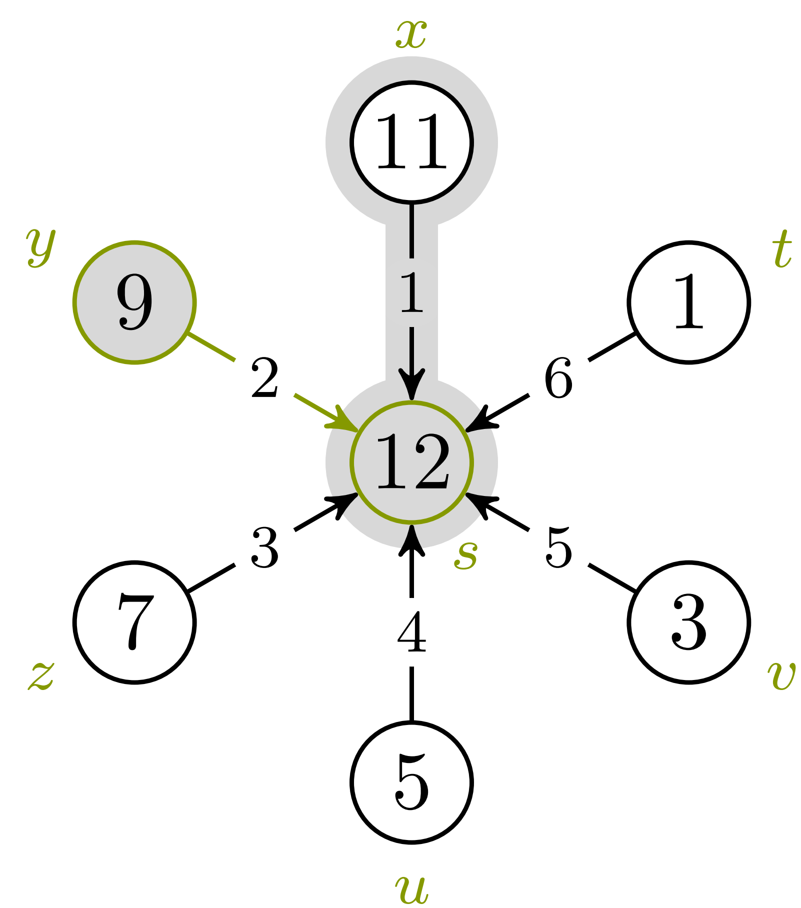
1 Relax(x, s)
2 Relax(y, s)
3 Relax(z, s)
4 Relax(u, s)
5 Relax(v, s)
6 Relax(t, s)



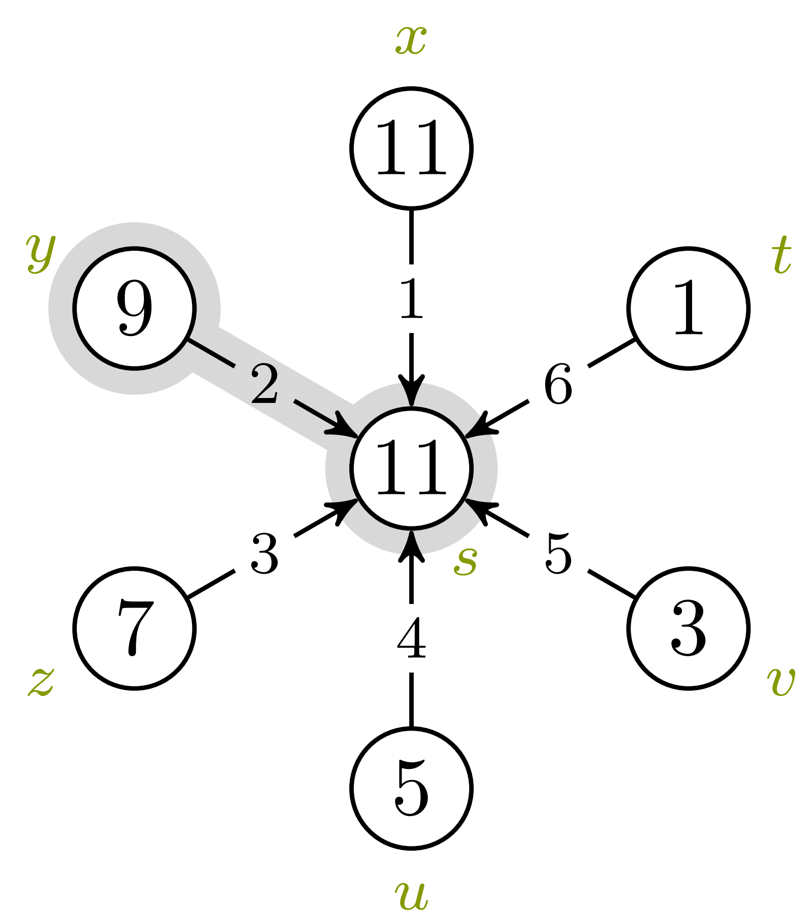
1 RELAX(x, s)
2 RELAX(y, s)
3 RELAX(z, s)
4 RELAX(u, s)
5 RELAX(v, s)
6 RELAX(t, s)



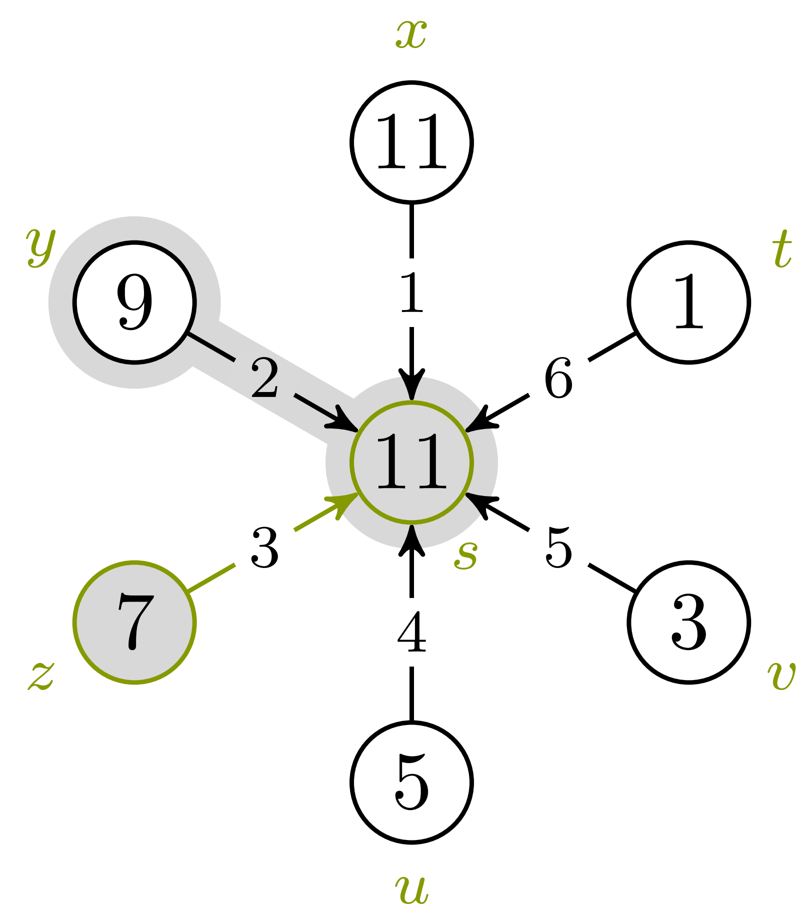
```
Relax(x, s)
Relax(y, s)
Relax(z, s)
Relax(u, s)
Relax(v, s)
Relax(t, s)
```



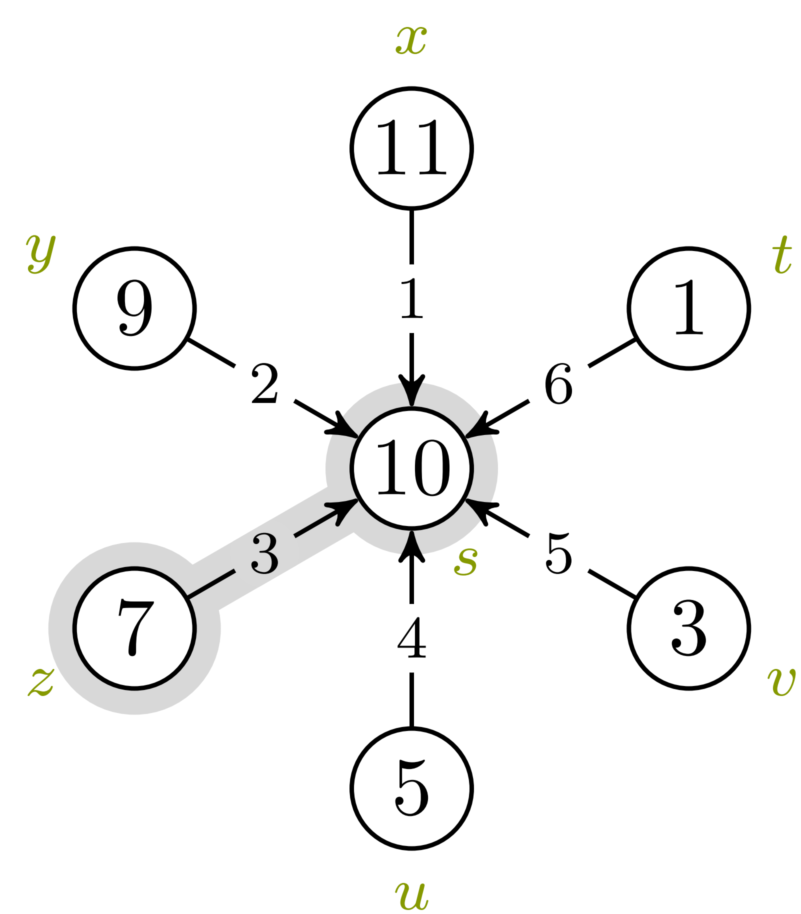
1 $\textsc{Relax}(x, s)$
2 $\textsc{Relax}(y, s)$
3 $\textsc{Relax}(z, s)$
4 $\textsc{Relax}(u, s)$
5 $\textsc{Relax}(v, s)$
6 $\textsc{Relax}(t, s)$



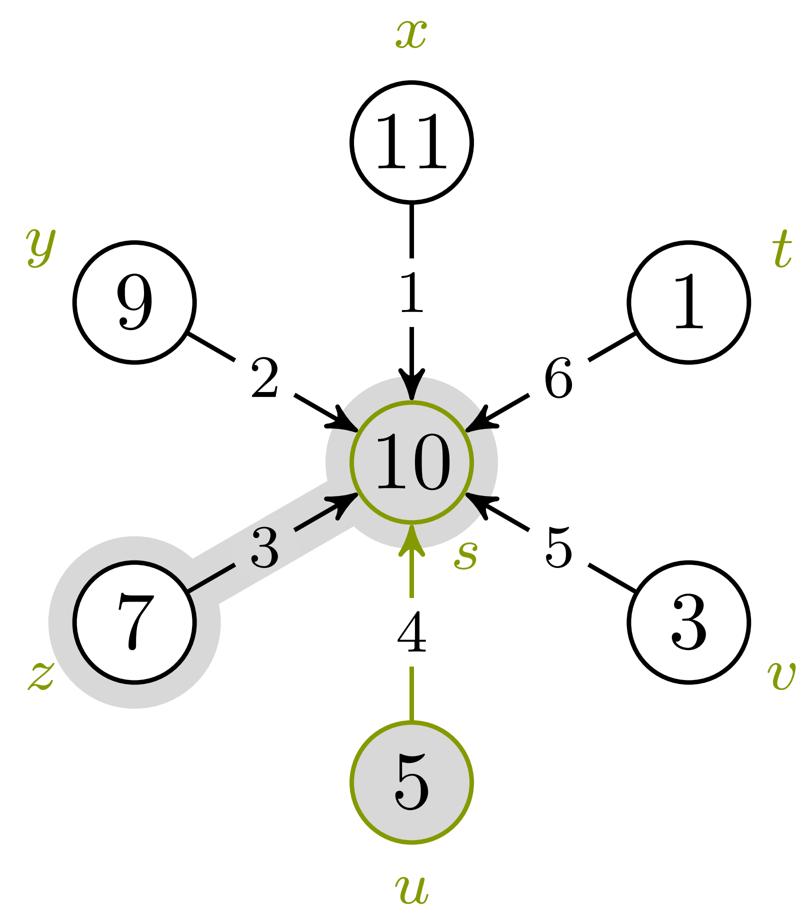
```
Relax(x, s)
Relax(y, s)
Relax(z, s)
Relax(u, s)
Relax(v, s)
Relax(t, s)
```



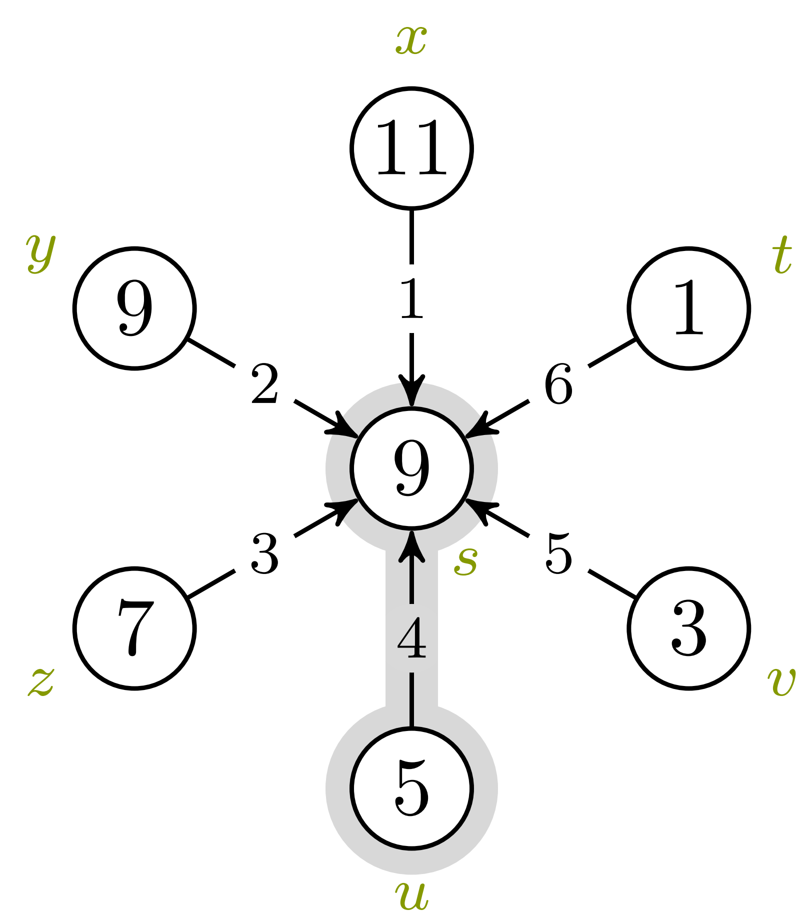
1 Relax(x, s)
2 Relax(y, s)
3 Relax(z, s)
4 Relax(u, s)
5 Relax(v, s)
6 Relax(t, s)



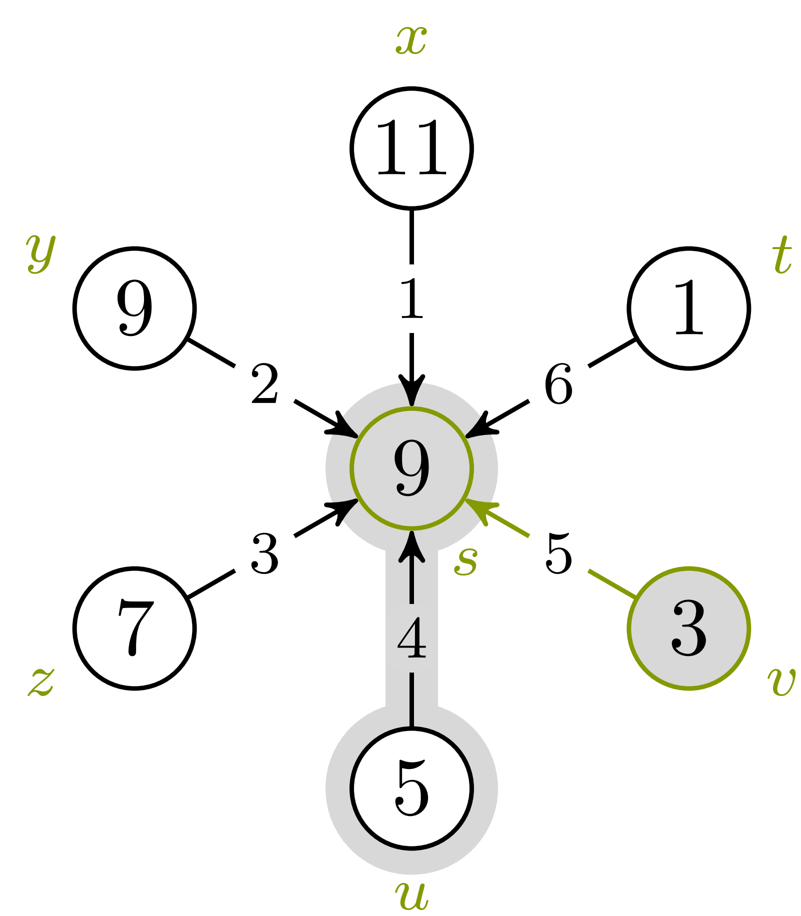
```
RELAX(x, s)
RELAX(y, s)
RELAX(z, s)
RELAX(u, s)
RELAX(v, s)
RELAX(t, s)
```



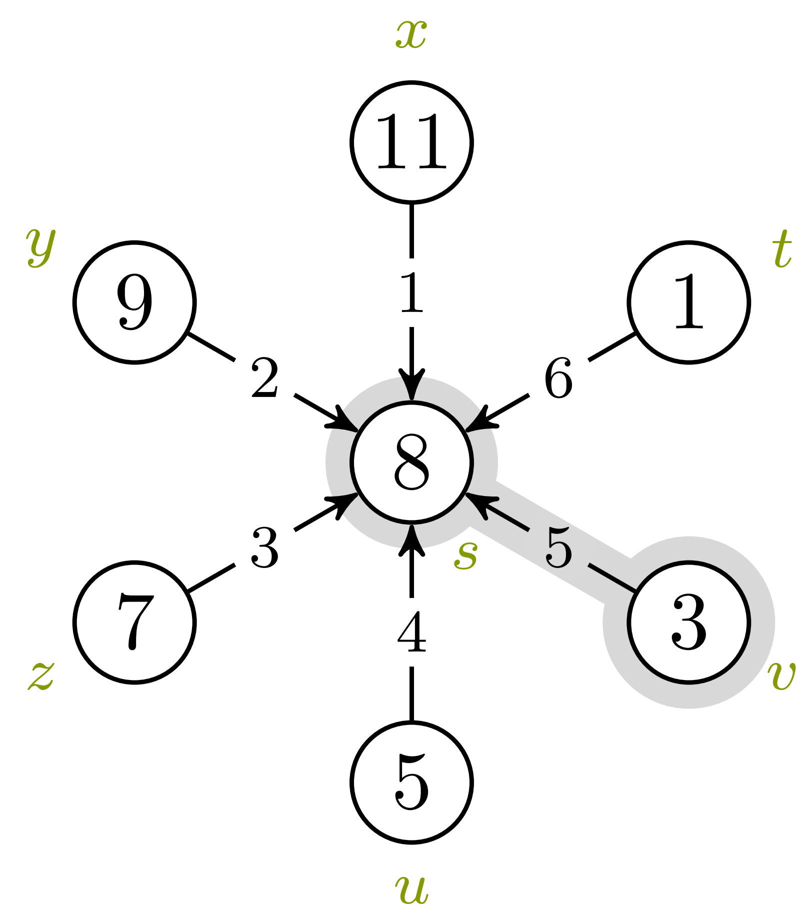
1 Relax(x, s)
2 Relax(y, s)
3 Relax(z, s)
4 Relax(u, s)
5 Relax(v, s)
6 Relax(t, s)



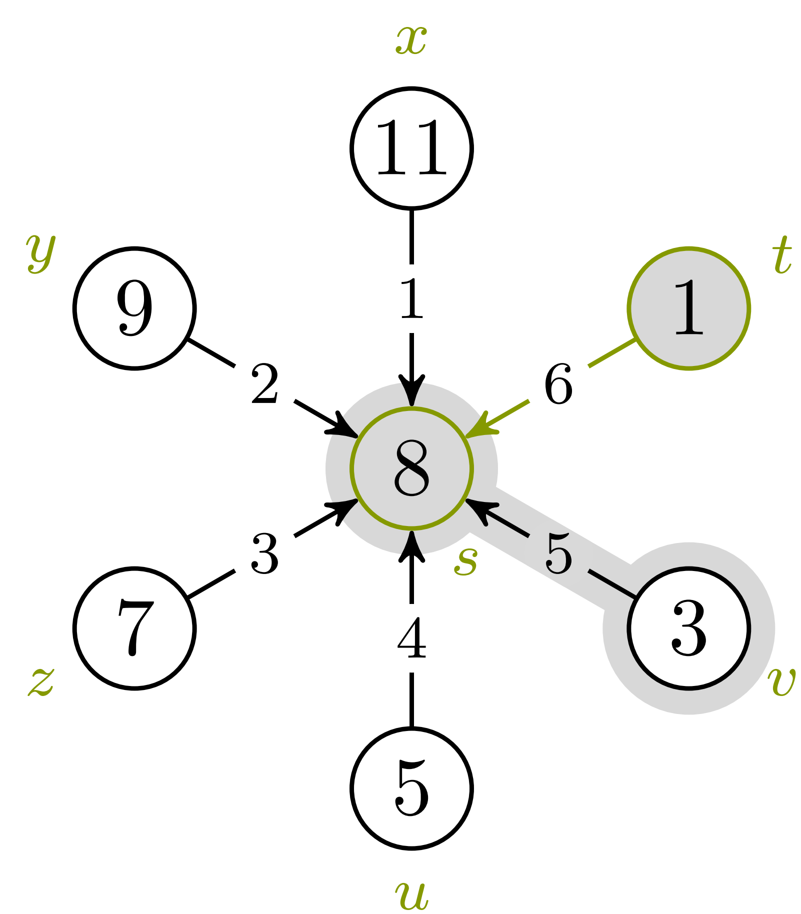
1 Relax(x, s)
2 Relax(y, s)
3 Relax(z, s)
4 Relax(u, s)
5 Relax(v, s)
6 Relax(t, s)



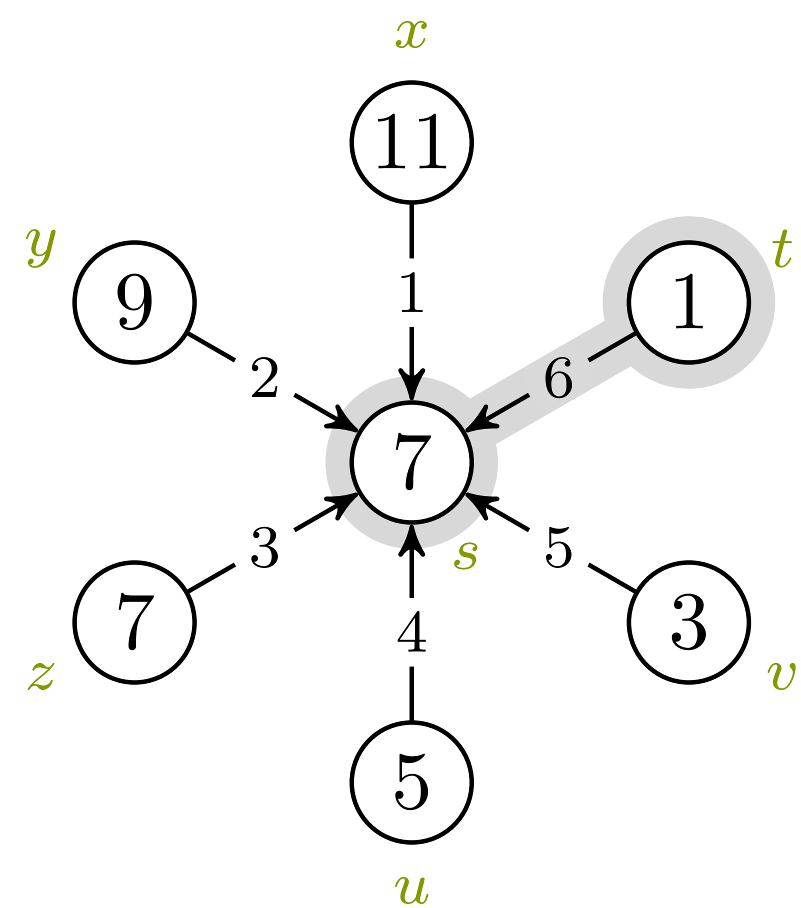
1 $\text{Relax}(x, s)$
2 $\text{Relax}(y, s)$
3 $\text{Relax}(z, s)$
4 $\text{Relax}(u, s)$
5 $\text{Relax}(v, s)$
6 $\text{Relax}(t, s)$



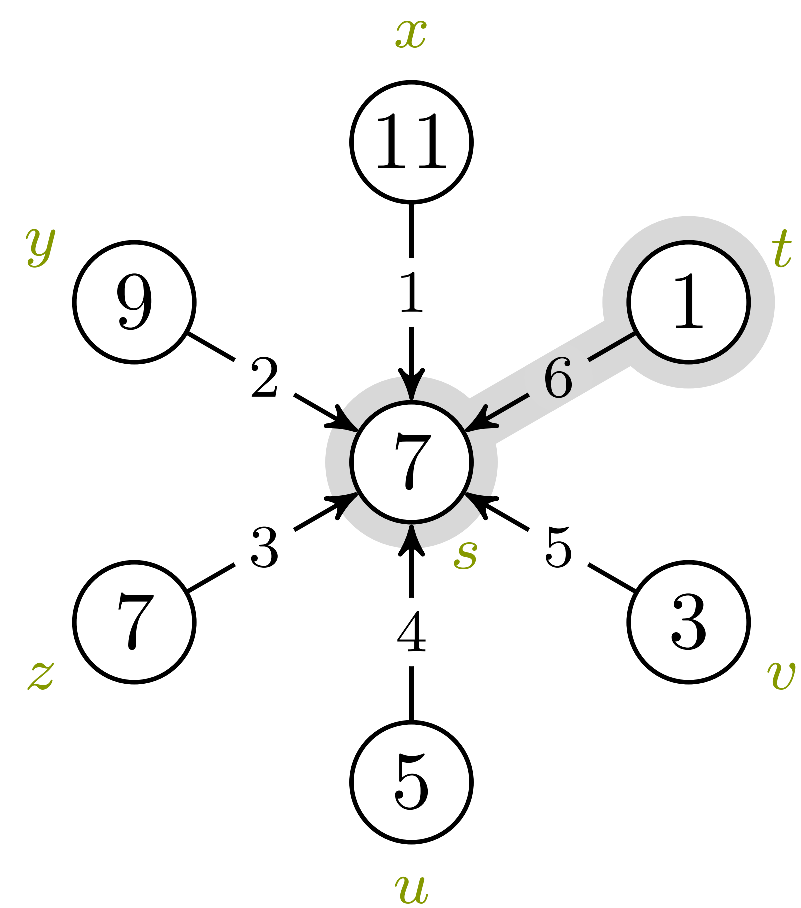
```
Relax(x, s)
Relax(y, s)
Relax(z, s)
Relax(u, s)
Relax(v, s)
Relax(t, s)
```



```
Relax(x, s)
Relax(y, s)
Relax(z, s)
Relax(u, s)
Relax(v, s)
Relax(t, s)
```

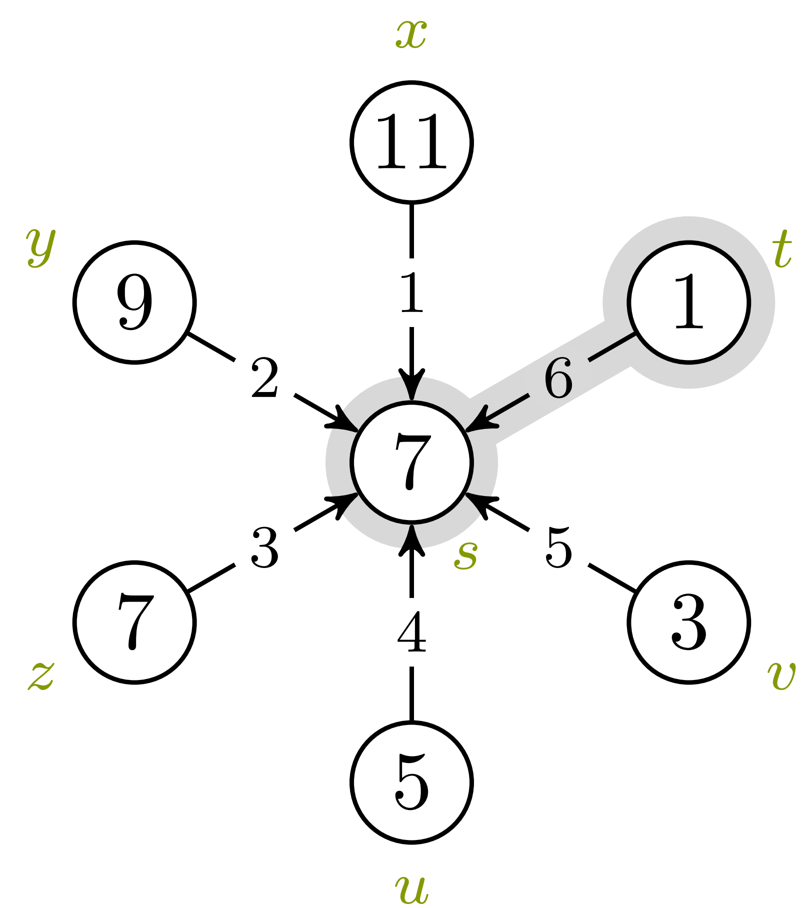


```
Relax(x, s)
Relax(y, s)
Relax(z, s)
Relax(u, s)
Relax(v, s)
Relax(t, s)
```

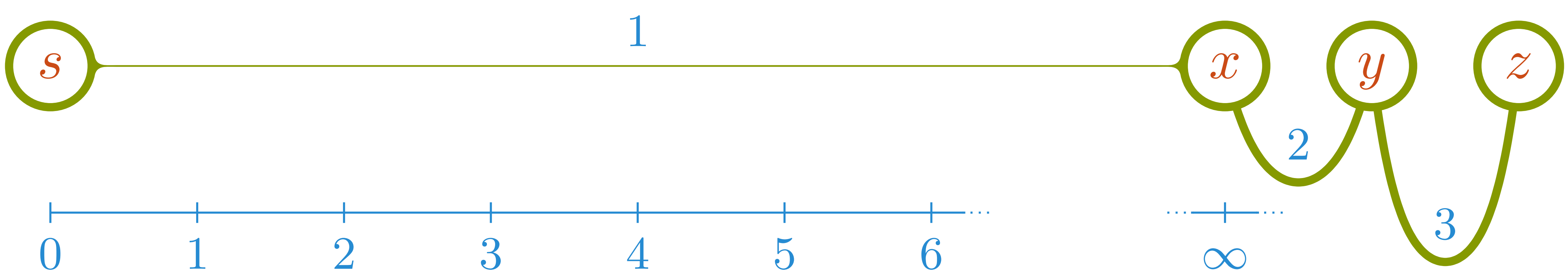


$s.d$ er min. over inn-kanter

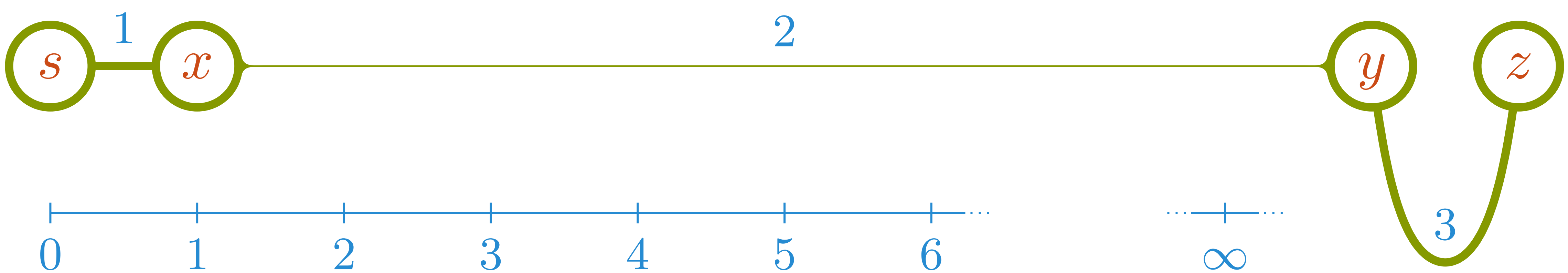
RELAX(x, s)
RELAX(y, s)
RELAX(z, s)
RELAX(u, s)
RELAX(v, s)
RELAX(t, s)



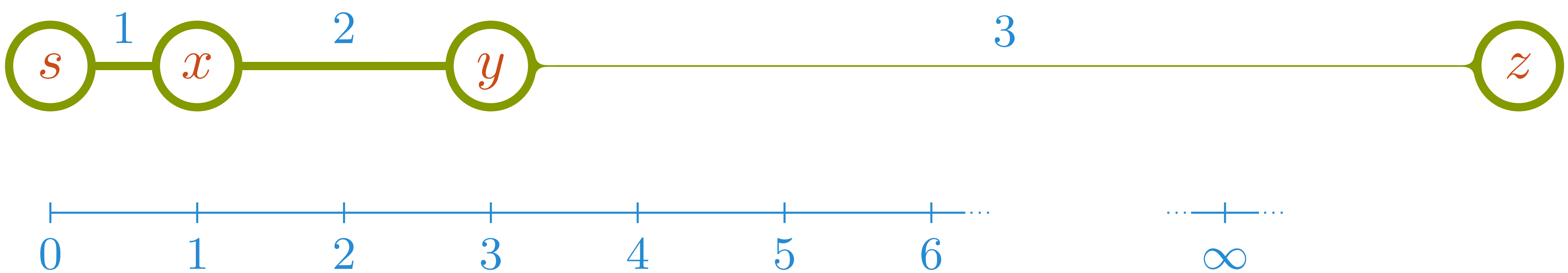
$x.d, y.d \dots$ rett $\implies$ $s.d$ rett

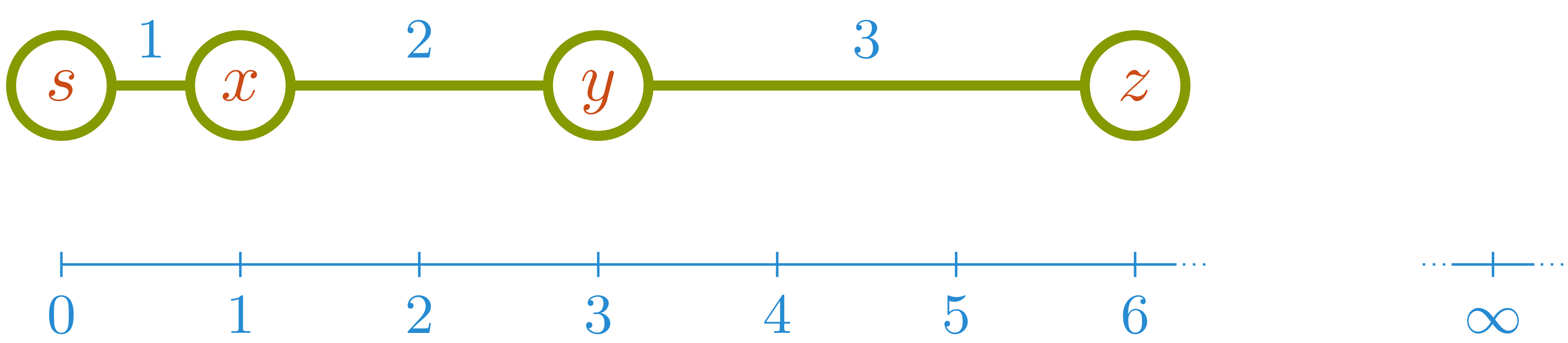


For å sikre oss mot «juks» starter vi med uendelige estimater



Vi «reparerer» så ett og ett estimat . . .


1
2
3
s
x
y
z
0
1
2
3
4
5
6
∞



. . . helt til alle er korrekte

Sti-slakkings-egenskapen

Sti-slakkings-egenskapen

Om p er en kortest vei fra s til v

Sti-slakkings-egenskapen

Om p er en kortest vei fra s til v og vi slakker kantene til p i rekkefølge,

Sti-slakkings-egenskapen

Om p er en kortest vei fra s til v og vi slakker kantene til p i rekkefølge, så vil v få riktig avstandsestimat.

Sti-slakkings-egenskapen

Om p er en kortest vei fra s til v og vi slakker kantene til p i rekkefølge, så vil v få riktig avstandsestimat. Det gjelder uavhengig av om andre slakkinger forekommer,

Sti-slakkings-egenskapen

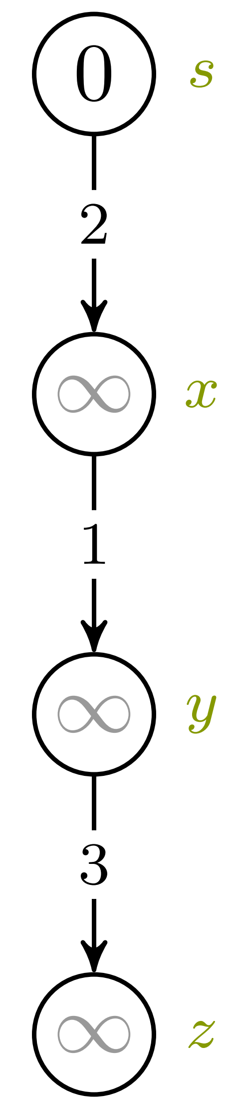
Om p er en kortest vei fra s til v og vi slakker kantene til p i rekkefølge, så vil v få riktig avstandsestimat. Det gjelder uavhengig av om andre slakkinger forekommer, selv om de kommer innimellom.

Sti-slakkings-egenskapen

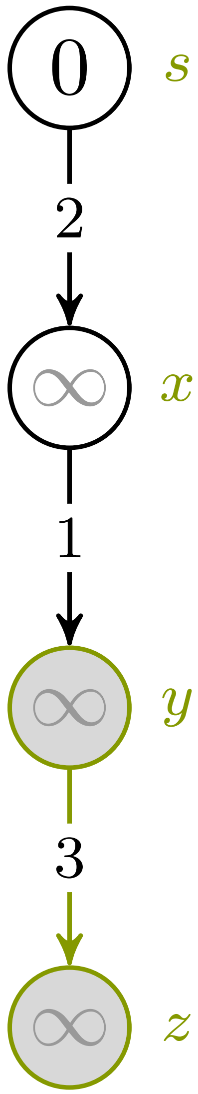
Om p er en kortest vei fra s til v og vi slakker kantene til p i rekkefølge, så vil v få riktig avstandsestimat. Det gjelder uavhengig av om andre slakkinger forekommer, selv om de kommer innimellom.

«The path-relaxation property»; flere slektninger på s. 650

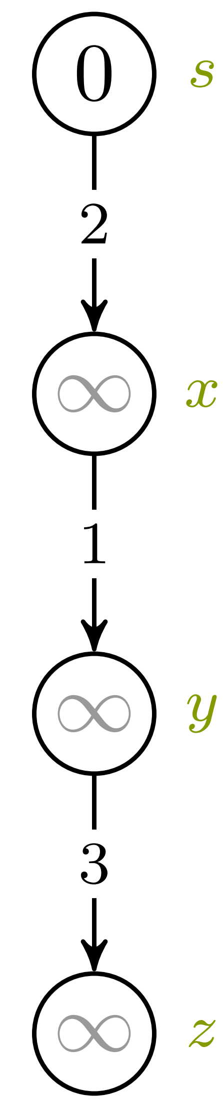
$\text{Relax}(y, z)$
$\text{Relax}(x, y)$
$\text{Relax}(s, x)$
$\text{Relax}(y, z)$
$\text{Relax}(x, y)$
$\text{Relax}(y, z)$
$\text{Relax}(s, x)$



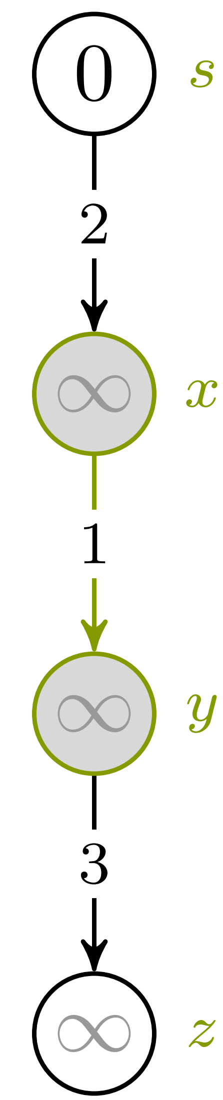
1 Relax(y, z)
2 Relax(x, y)
3 Relax(s, x)
4 Relax(y, z)
5 Relax(x, y)
6 Relax(y, z)
7 Relax(s, x)



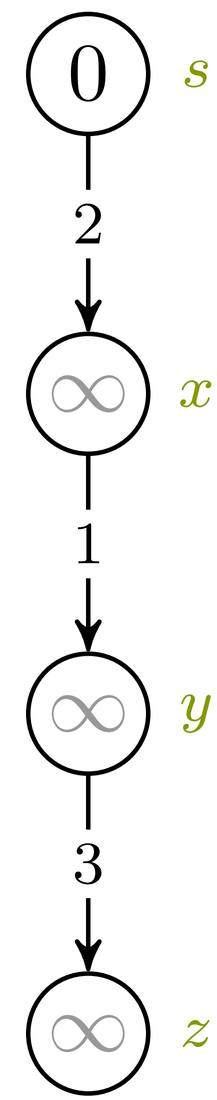
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



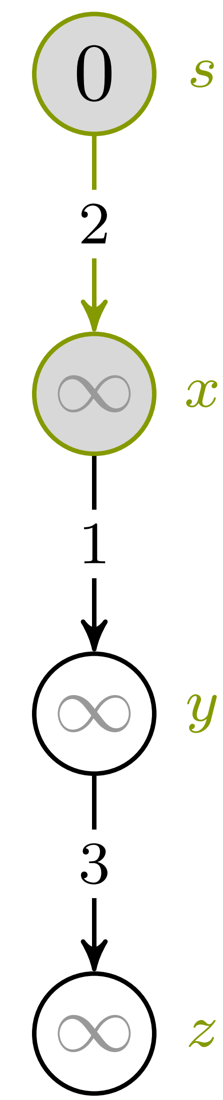
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



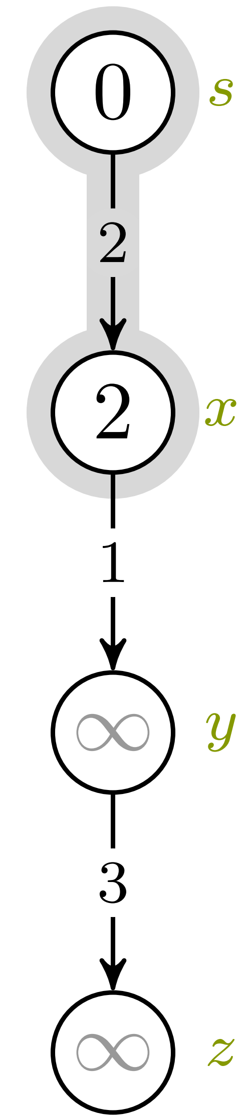
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



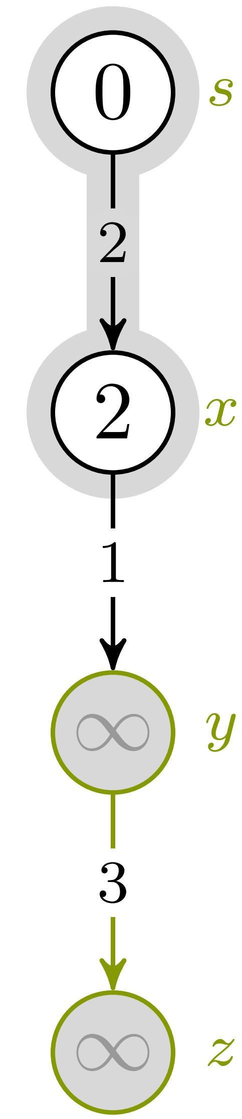
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



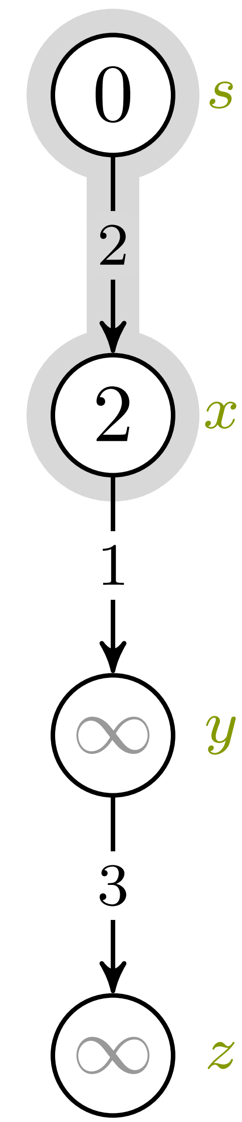
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



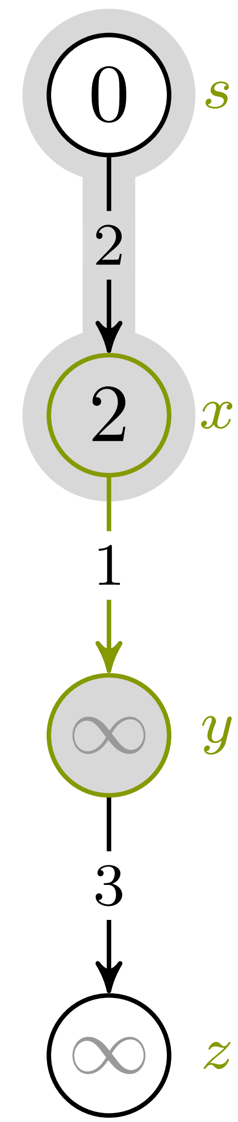
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



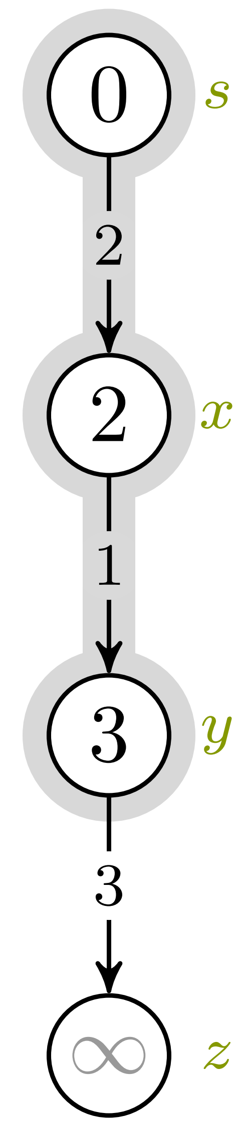
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)



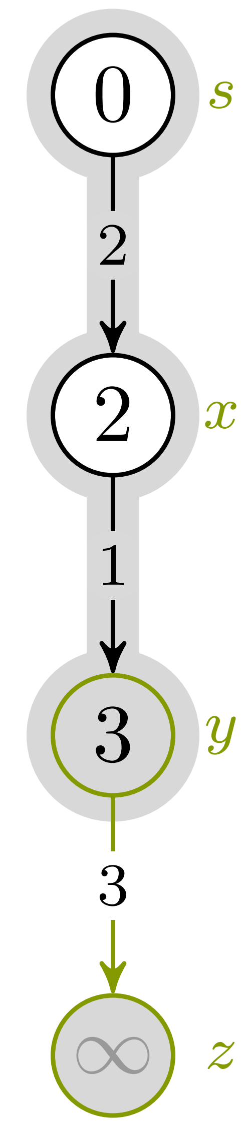
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



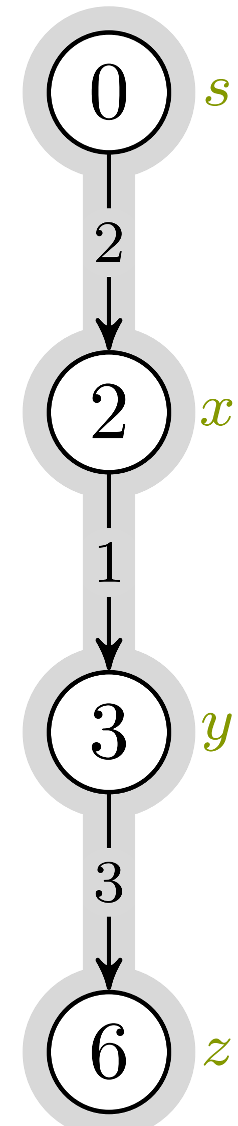
$\textsc{Relax}(y, z)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(s, x)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(s, x)$



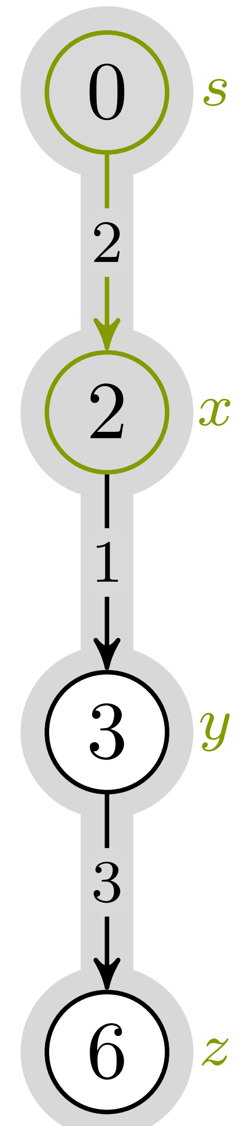
1. $\textsc{Relax}(y, z)$
2. $\textsc{Relax}(x, y)$
3. $\textsc{Relax}(s, x)$
4. $\textsc{Relax}(y, z)$
5. $\textsc{Relax}(x, y)$
6. $\textsc{Relax}(y, z)$
7. $\textsc{Relax}(s, x)$



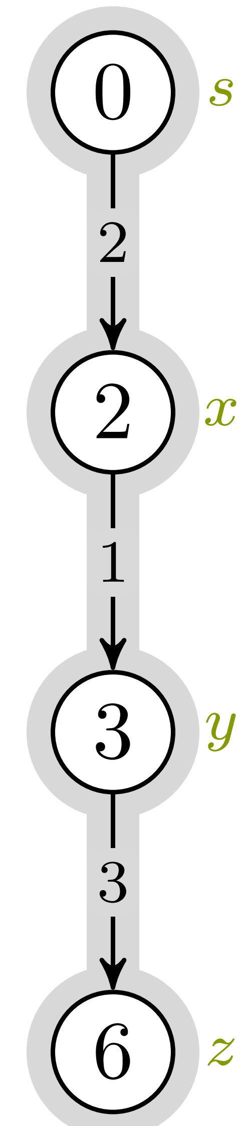
$\textsc{Relax}(y, z)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(s, x)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(s, x)$



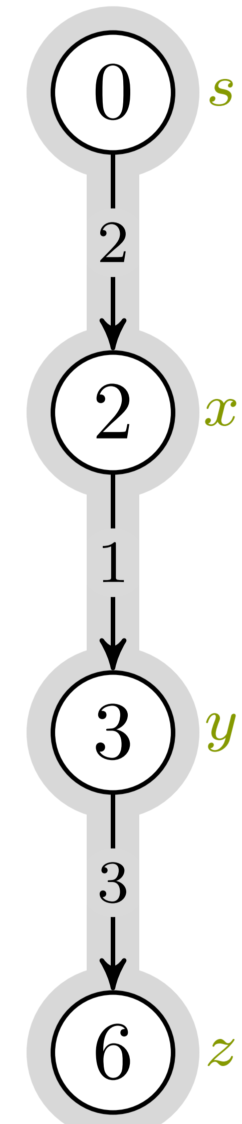
```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```



```
Relax(y, z)
Relax(x, y)
Relax(s, x)
Relax(y, z)
Relax(x, y)
Relax(y, z)
Relax(s, x)
```

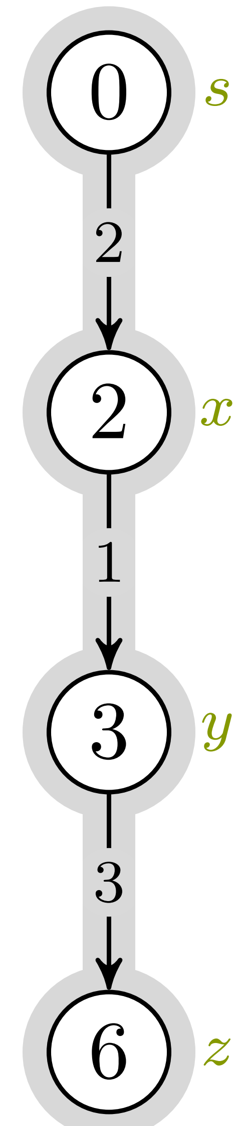


1 $\textsc{Relax}(y, z)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(s, x)$
4 $\textsc{Relax}(y, z)$
5 $\textsc{Relax}(x, y)$
6 $\textsc{Relax}(y, z)$
7 $\textsc{Relax}(s, x)$



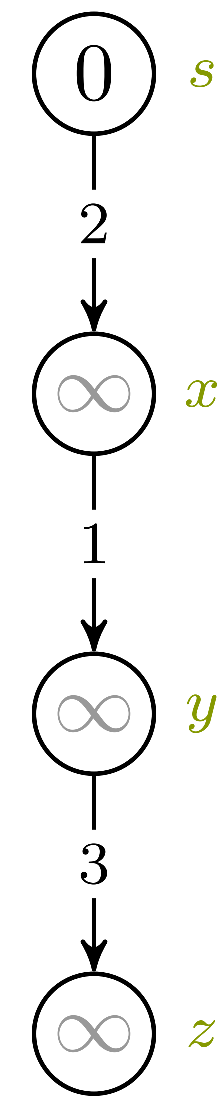
Flere kall var bortkastet

$\textsc{Relax}(y, z)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(s, x)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(s, x)$

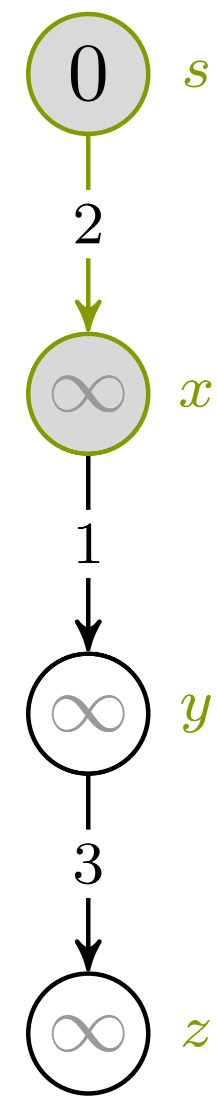


La oss droppe dem!

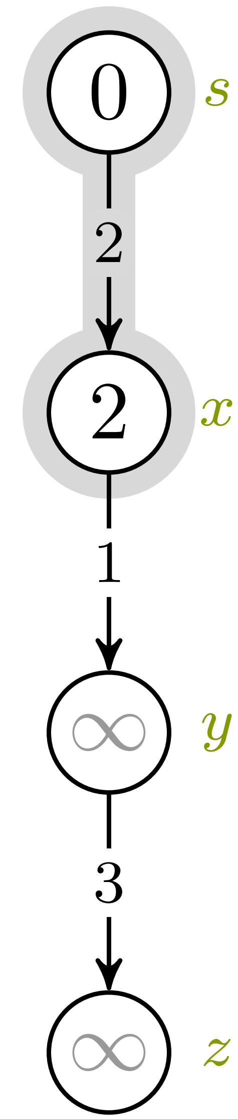
1. ~~RELAX(y, z)~~
2. ~~RELAX(x, y)~~
3. RELAX(s, x)
4. ~~RELAX(y, z)~~
5. RELAX(x, y)
6. RELAX(y, z)
7. ~~RELAX(s, x)~~



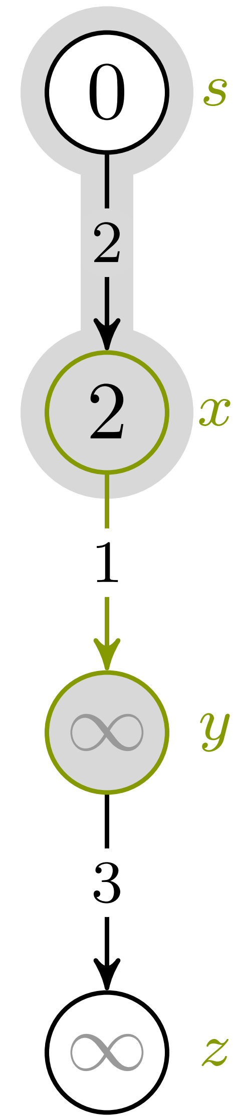
1. ~~RELAX(y, z)~~
2. ~~RELAX(x, y)~~
3. RELAX(s, x)
4. ~~RELAX(y, z)~~
5. RELAX(x, y)
6. RELAX(y, z)
7. ~~RELAX(s, x)~~



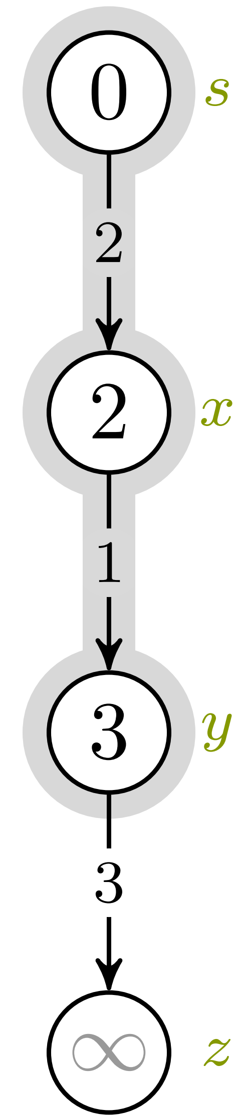
1. ~~RELAX(y, z)~~
2. ~~RELAX(x, y)~~
3. RELAX(s, x)
4. ~~RELAX(y, z)~~
5. RELAX(x, y)
6. RELAX(y, z)
7. ~~RELAX(s, x)~~



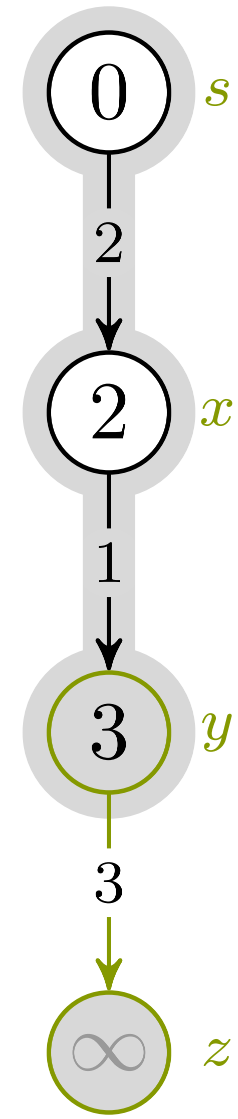
~~Relax(y, z)~~
~~Relax(x, y)~~
Relax(s, x)
~~Relax(y, z)~~
Relax(x, y)
Relax(y, z)
~~Relax(s, x)~~



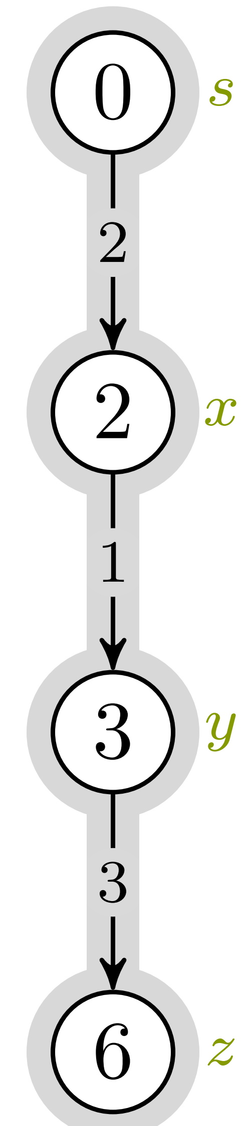
1 ~~RELAX(y, z)~~
2 ~~RELAX(x, y)~~
3 RELAX(s, x)
4 ~~RELAX(y, z)~~
5 RELAX(x, y)
6 RELAX(y, z)
7 ~~RELAX(s, x)~~



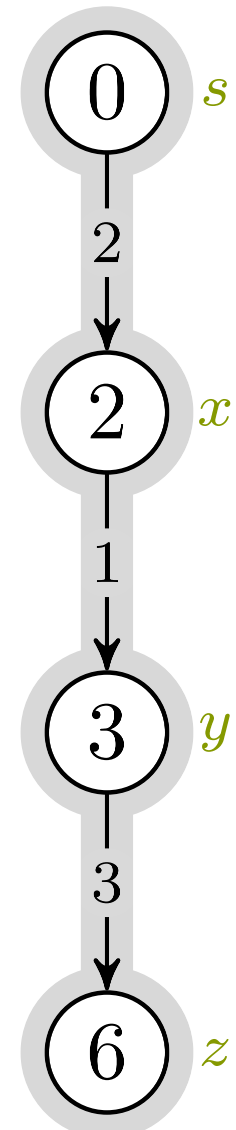
~~Relax(y, z)~~
~~Relax(x, y)~~
Relax(s, x)
~~Relax(y, z)~~
Relax(x, y)
Relax(y, z)
~~Relax(s, x)~~



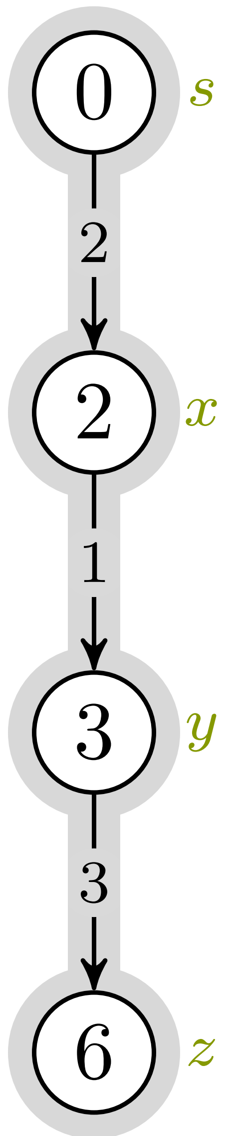
1. ~~RELAX(y, z)~~
2. ~~RELAX(x, y)~~
3. RELAX(s, x)
4. ~~RELAX(y, z)~~
5. RELAX(x, y)
6. RELAX(y, z)
7. ~~RELAX(s, x)~~



1 ~~$\textsc{Relax}(y, z)$~~
2 ~~$\textsc{Relax}(x, y)$~~
3 $\textsc{Relax}(s, x)$
4 ~~$\textsc{Relax}(y, z)$~~
5 $\textsc{Relax}(x, y)$
6 $\textsc{Relax}(y, z)$
7 ~~$\textsc{Relax}(s, x)$~~

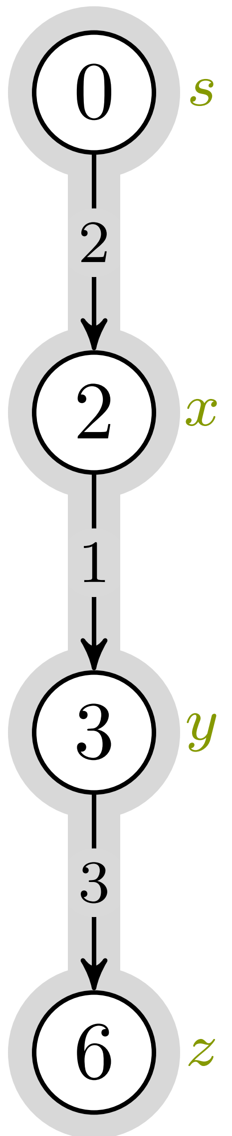


1 ~~RELAX(y, z)~~
2 ~~RELAX(x, y)~~
3 RELAX(s, x)
4 ~~RELAX(y, z)~~
5 RELAX(x, y)
6 RELAX(y, z)
7 ~~RELAX(s, x)~~



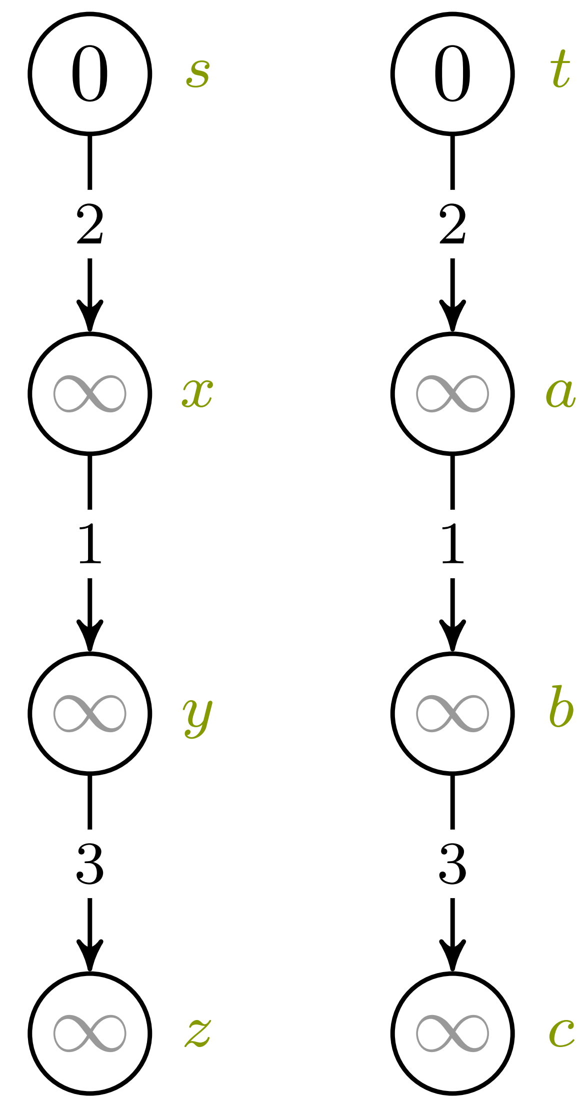
Samme utvikling; samme svar

1 ~~RELAX(y, z)~~
2 ~~RELAX(x, y)~~
3 RELAX(s, x)
4 ~~RELAX(y, z)~~
5 RELAX(x, y)
6 RELAX(y, z)
7 ~~RELAX(s, x)~~



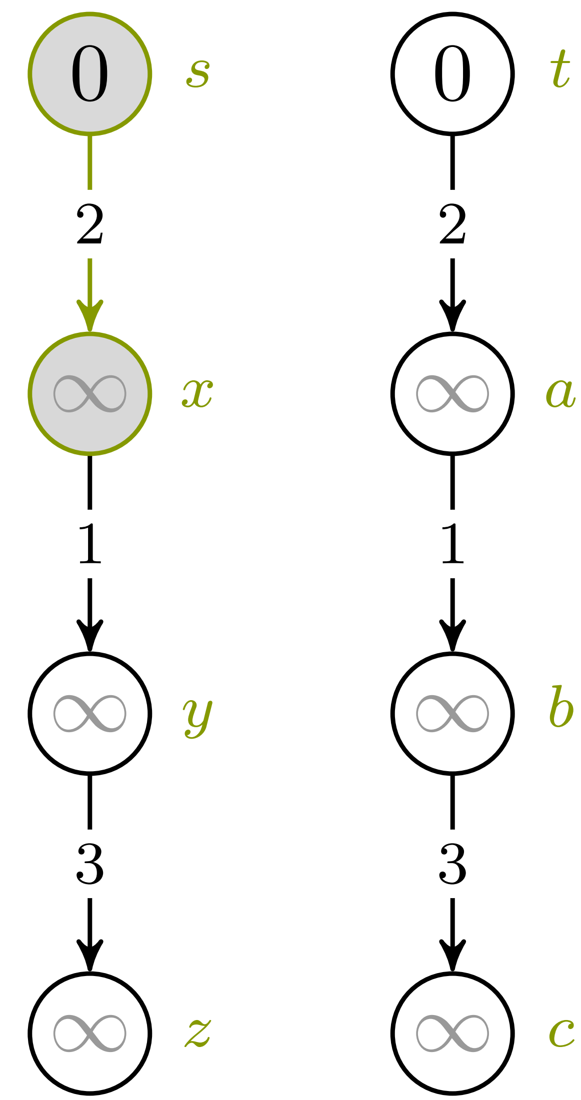
Følger rekkefølgen langs stien!

```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```

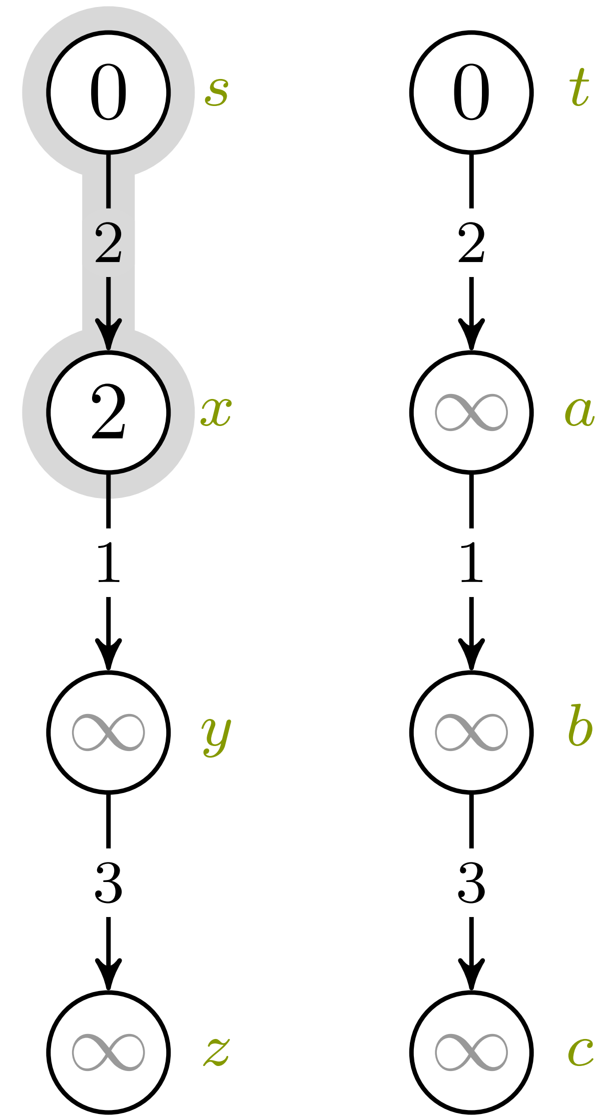


Slakk alle kanter én gang

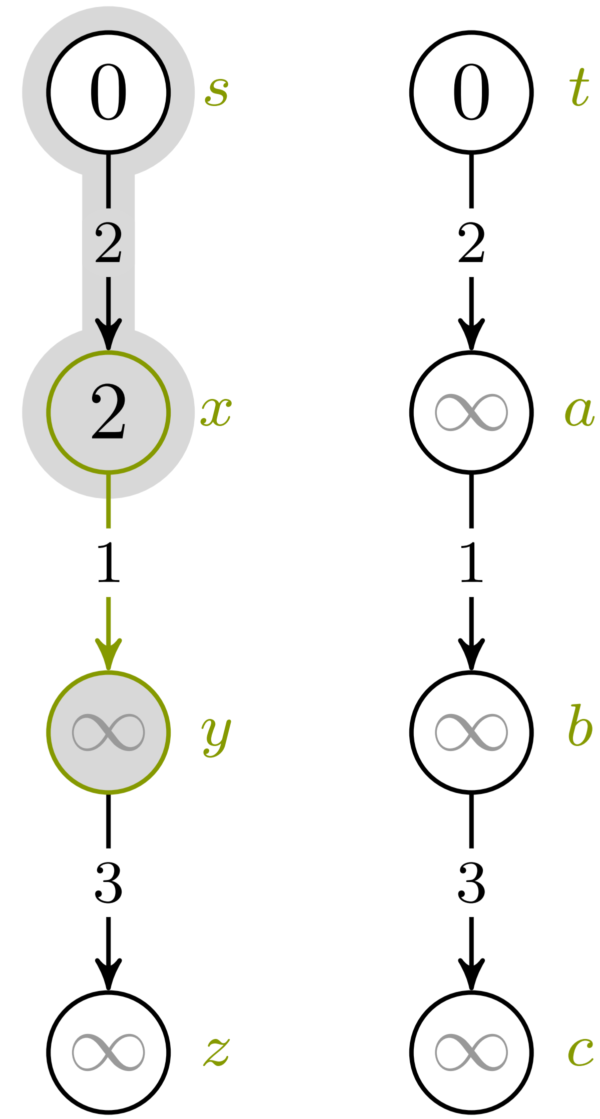
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



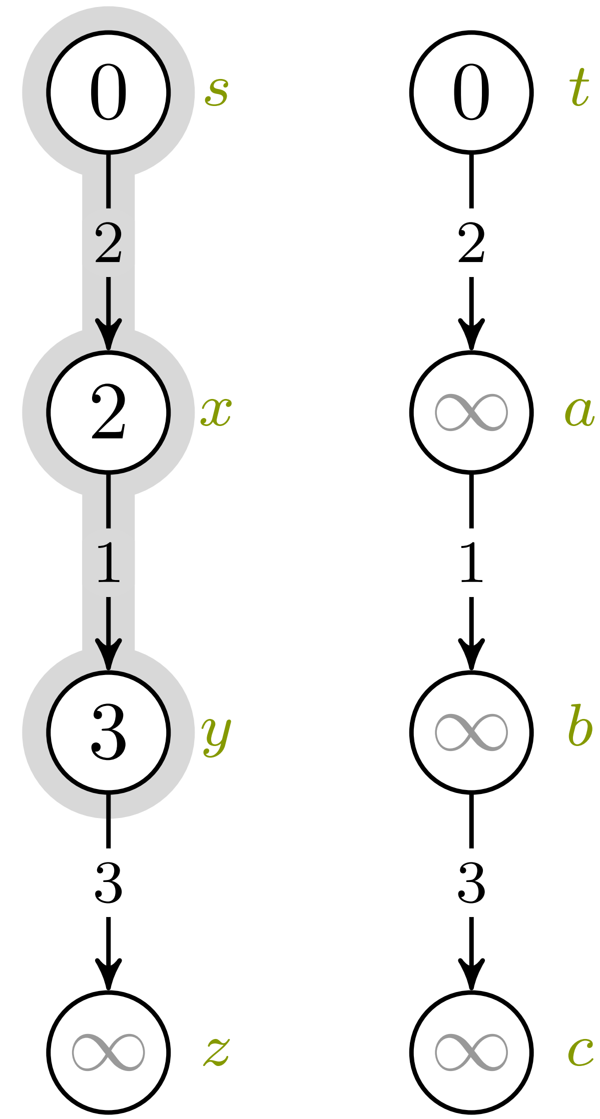
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)



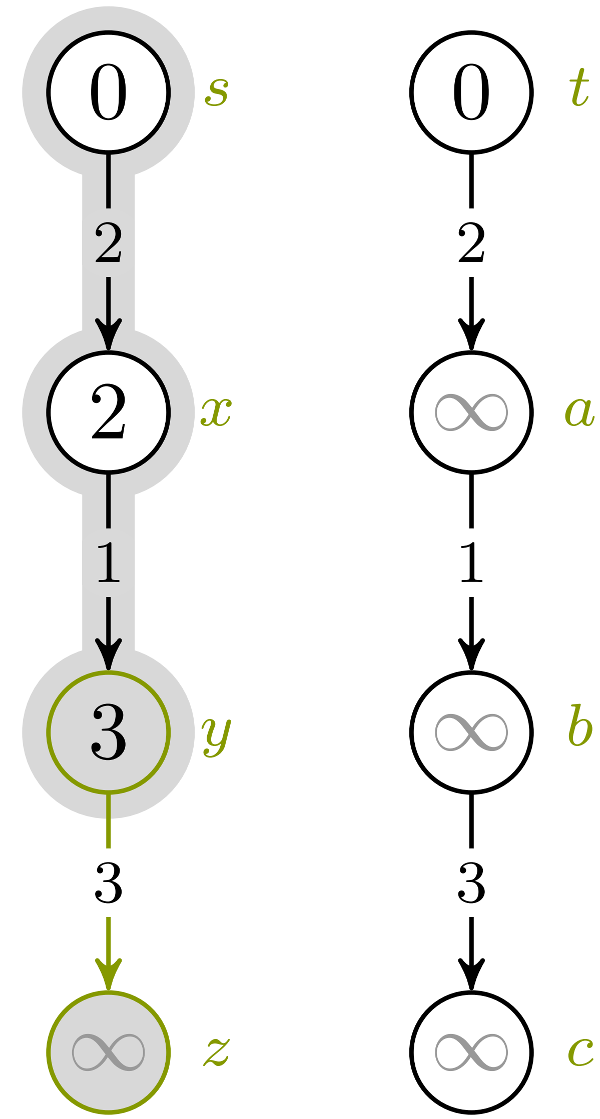
1 Relax(s, x)
2 Relax(x, y)
3 Relax(y, z)
4 Relax(b, c)
5 Relax(a, b)
6 Relax(t, a)



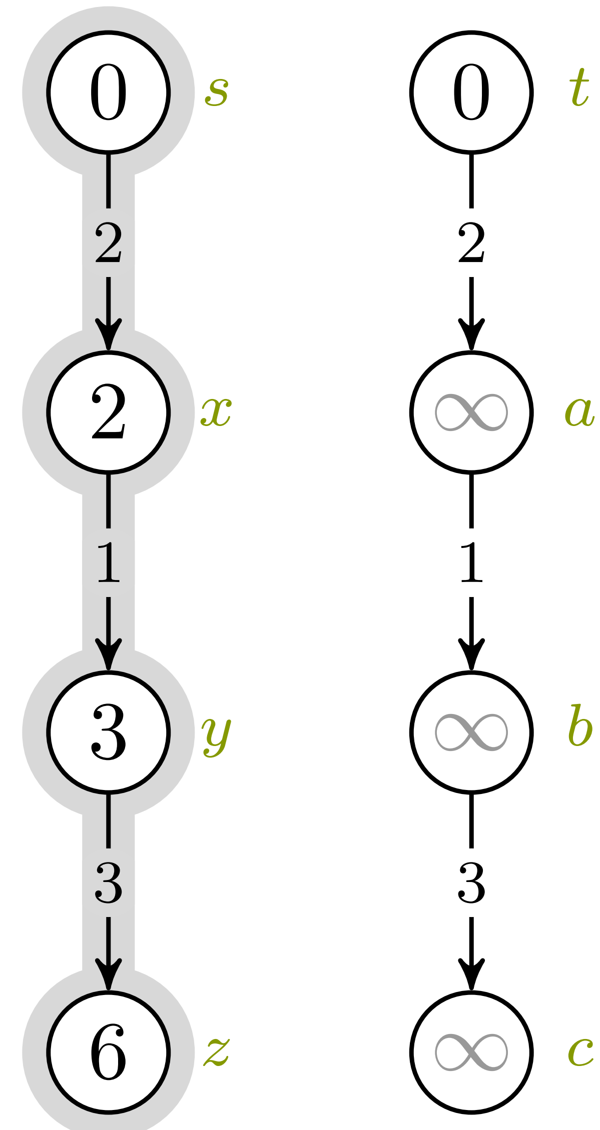
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



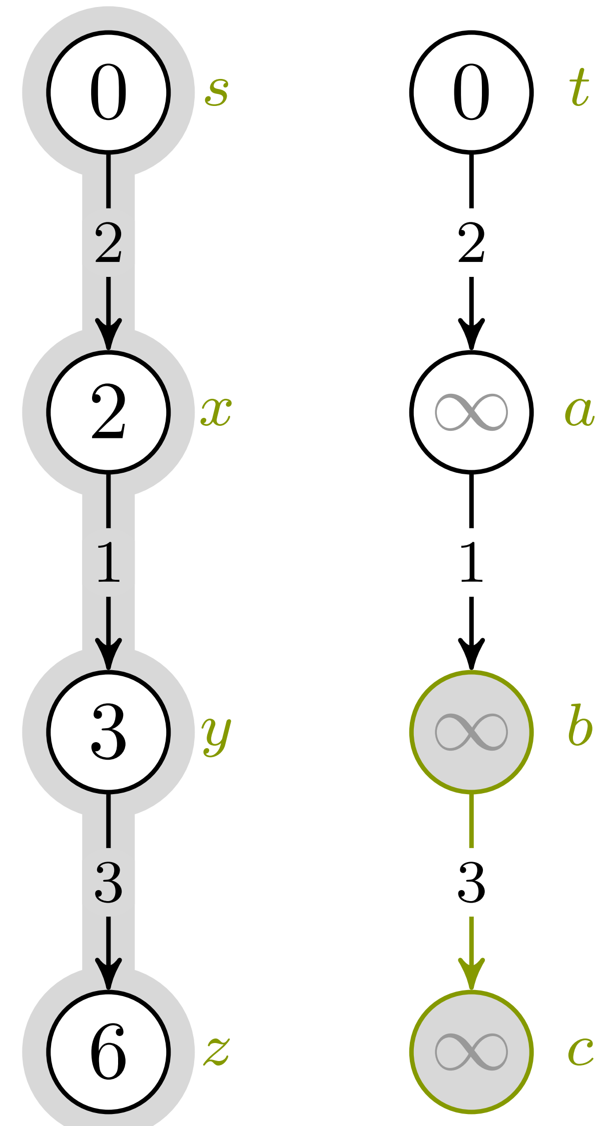
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



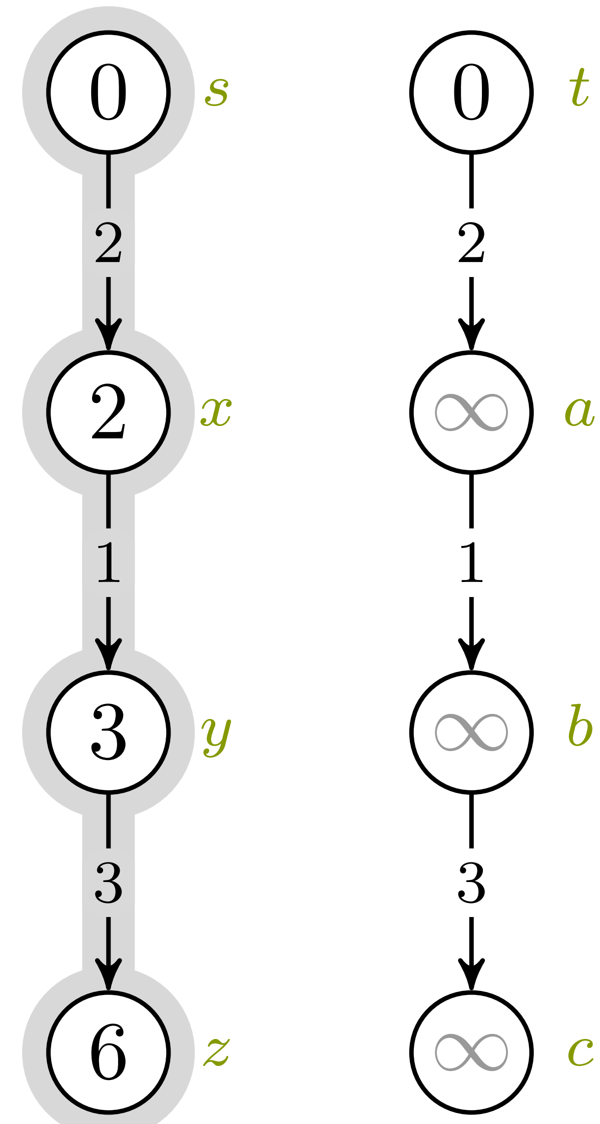
1 Relax(s, x)
2 Relax(x, y)
3 Relax(y, z)
4 Relax(b, c)
5 Relax(a, b)
6 Relax(t, a)



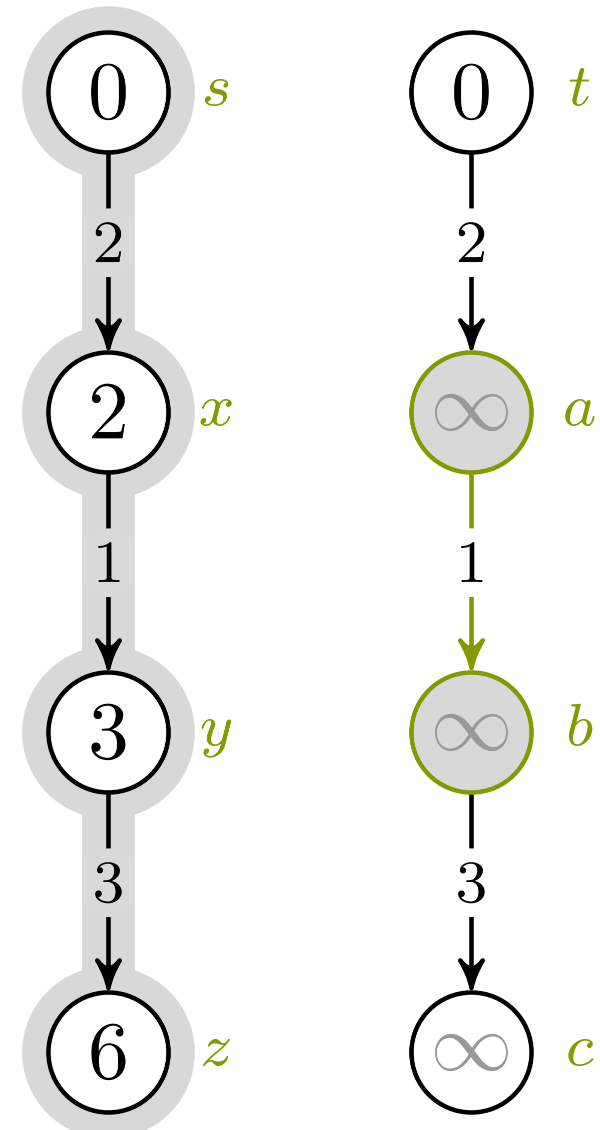
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)



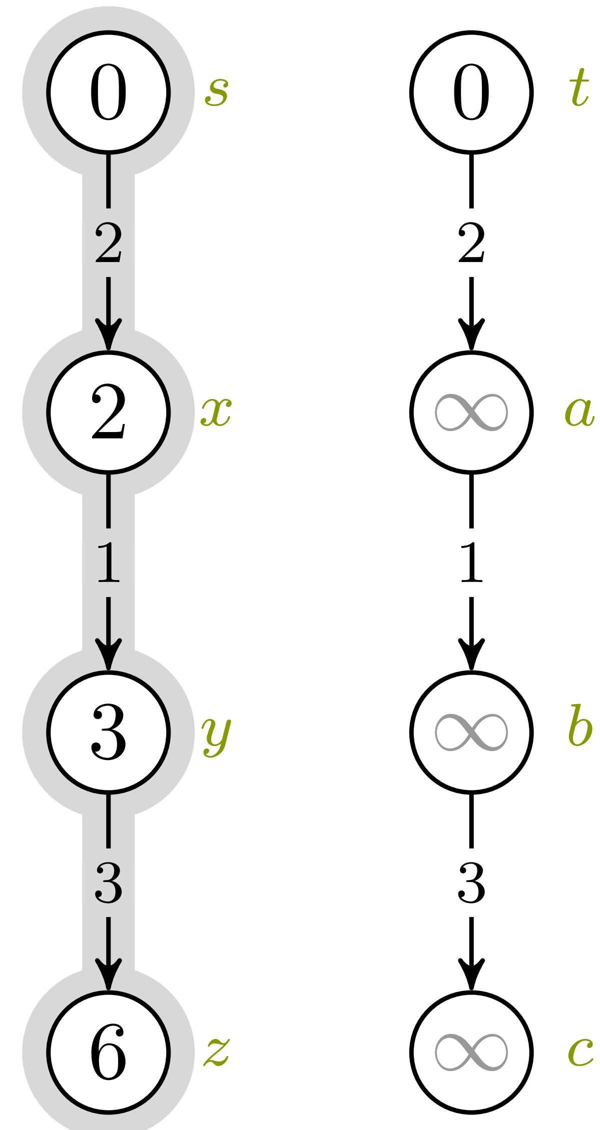
1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(y, z)$
4 $\textsc{Relax}(b, c)$
5 $\textsc{Relax}(a, b)$
6 $\textsc{Relax}(t, a)$



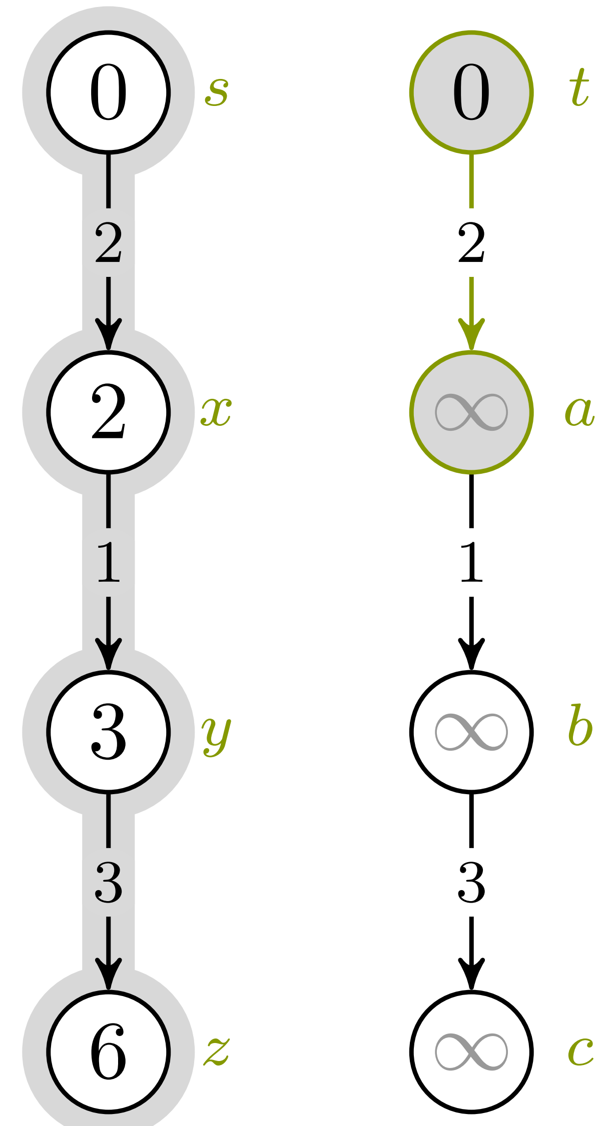
1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(y, z)$
4 $\textsc{Relax}(b, c)$
5 $\textsc{Relax}(a, b)$
6 $\textsc{Relax}(t, a)$



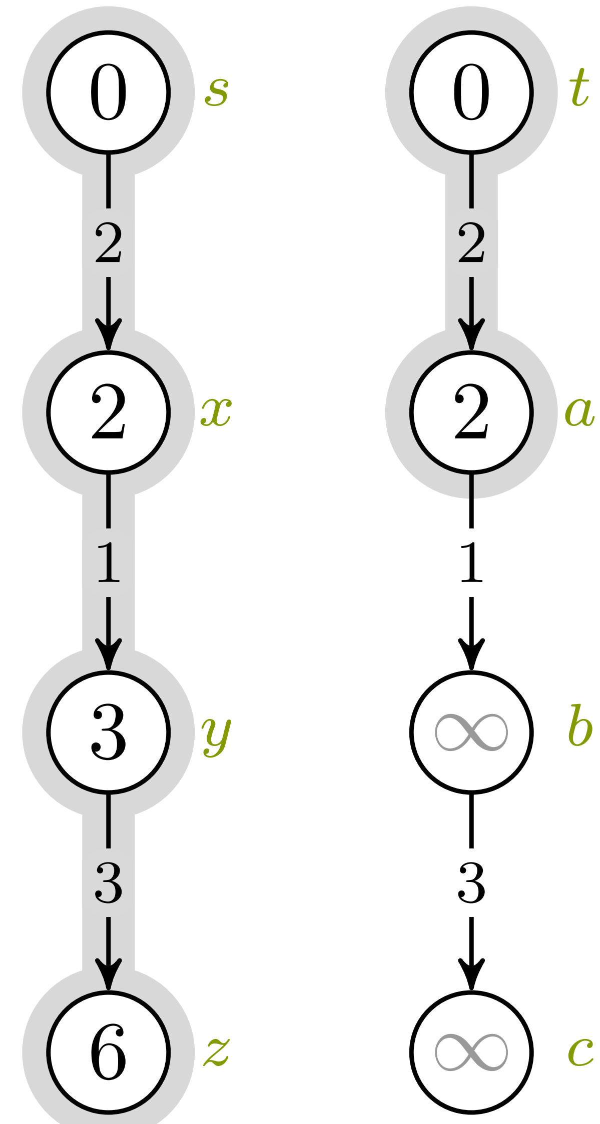
1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(y, z)$
4 $\textsc{Relax}(b, c)$
5 $\textsc{Relax}(a, b)$
6 $\textsc{Relax}(t, a)$



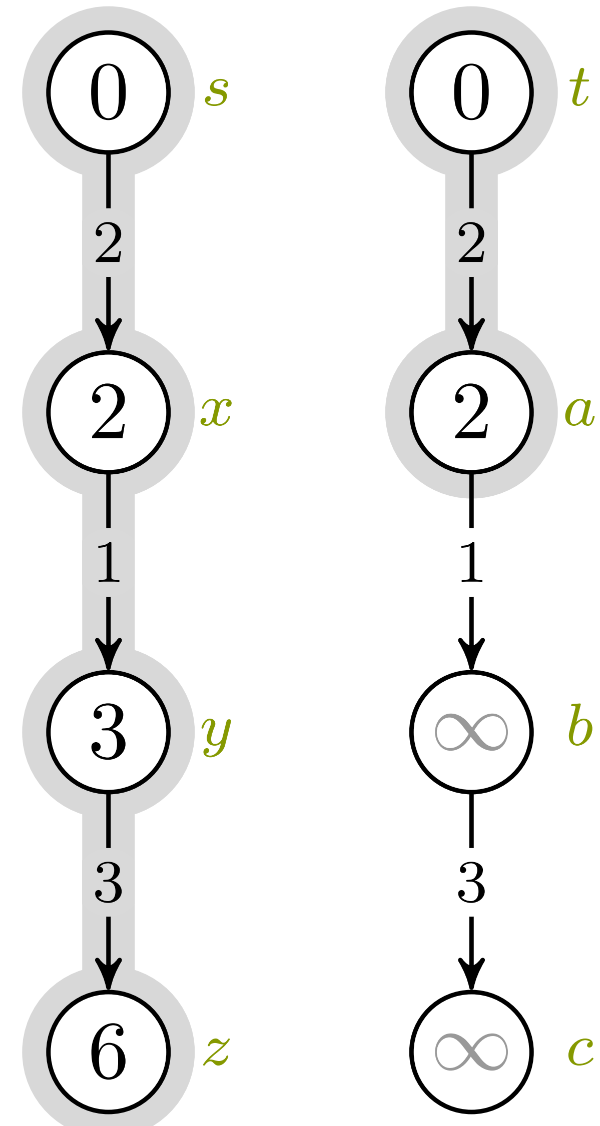
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```

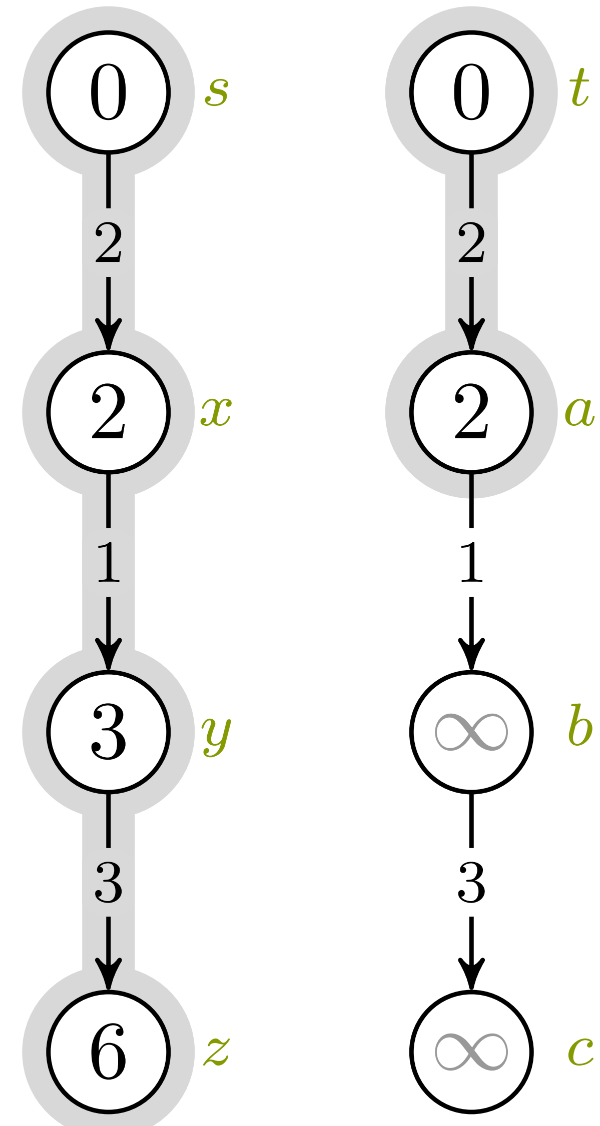


Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)



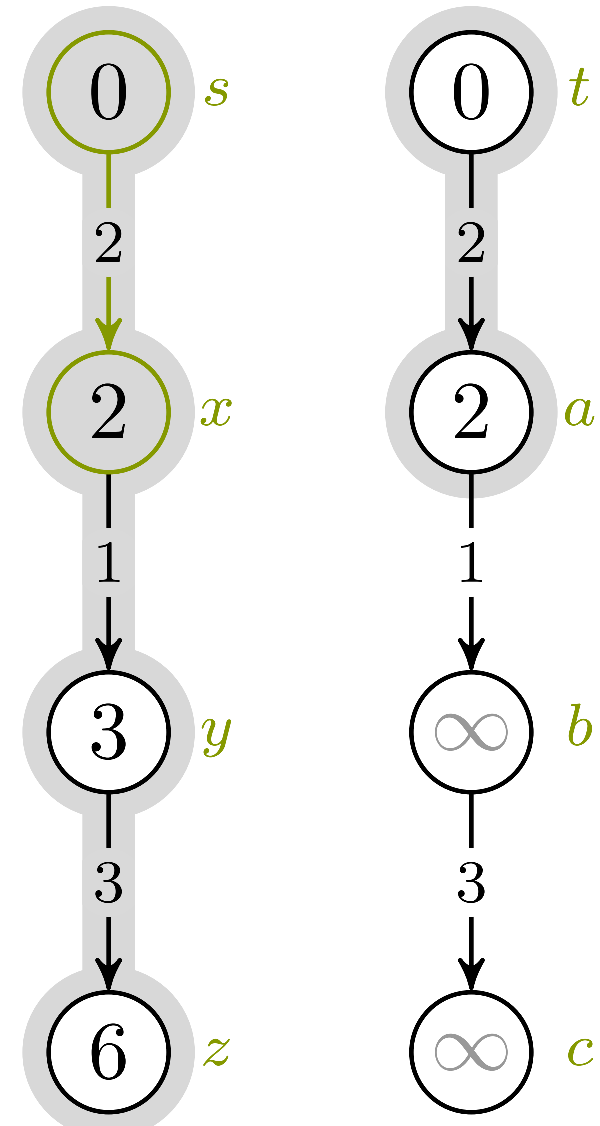
Flaks? Ferdig!

1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(y, z)$
4 $\textsc{Relax}(b, c)$
5 $\textsc{Relax}(a, b)$
6 $\textsc{Relax}(t, a)$

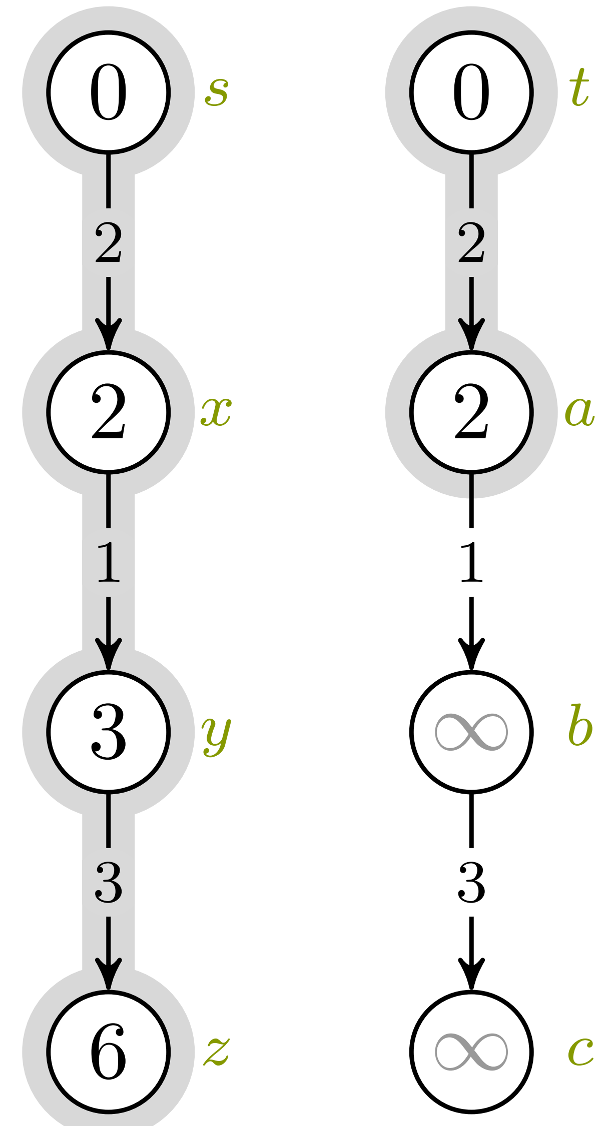


Uflaks? Ett hakk videre

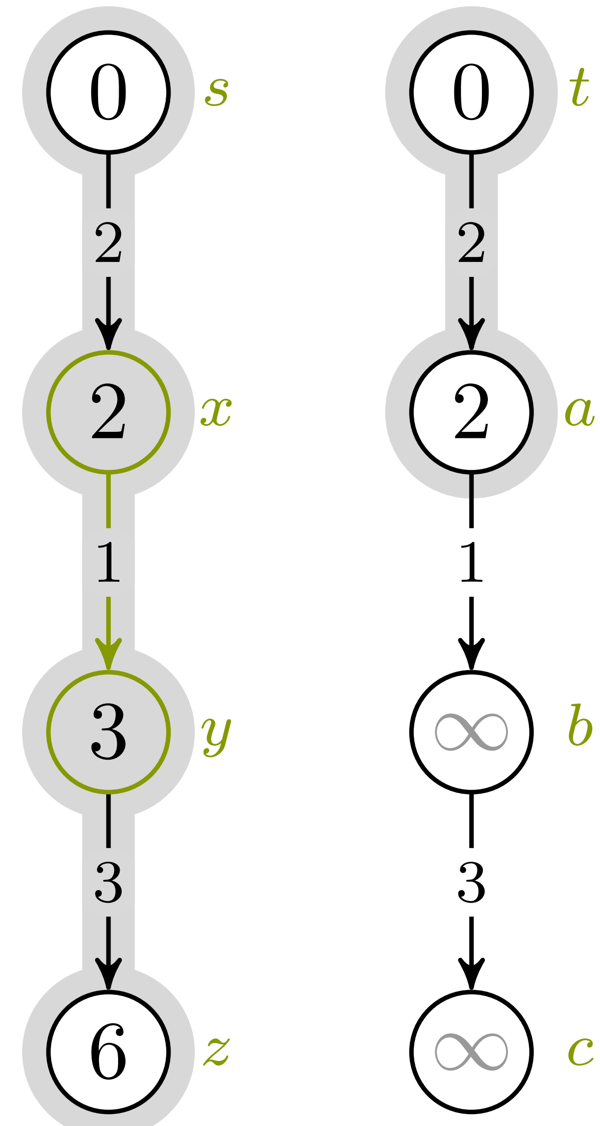
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



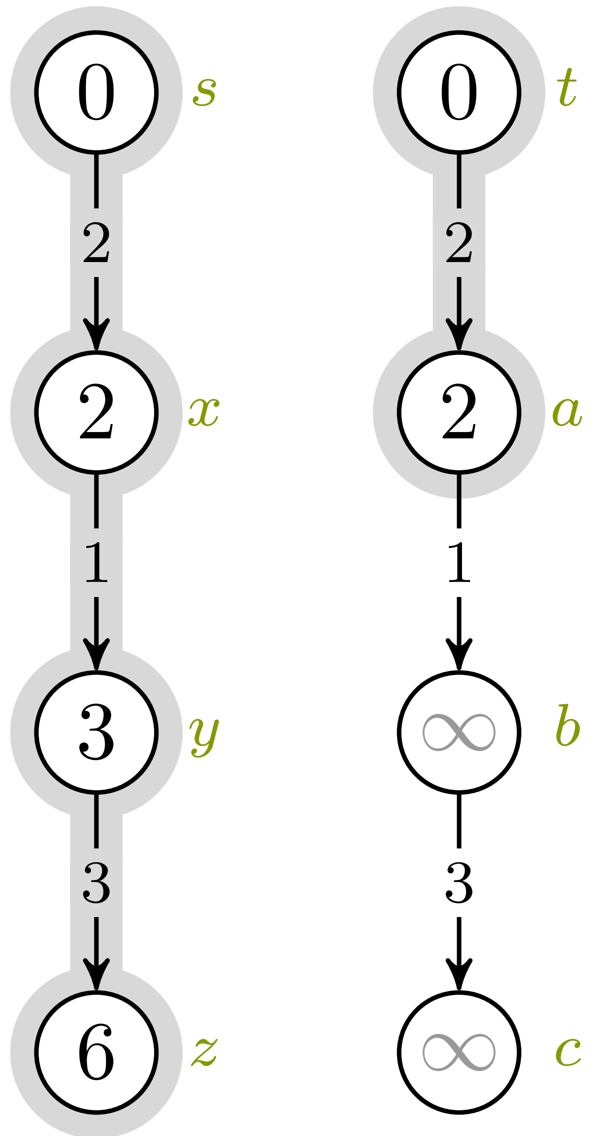
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



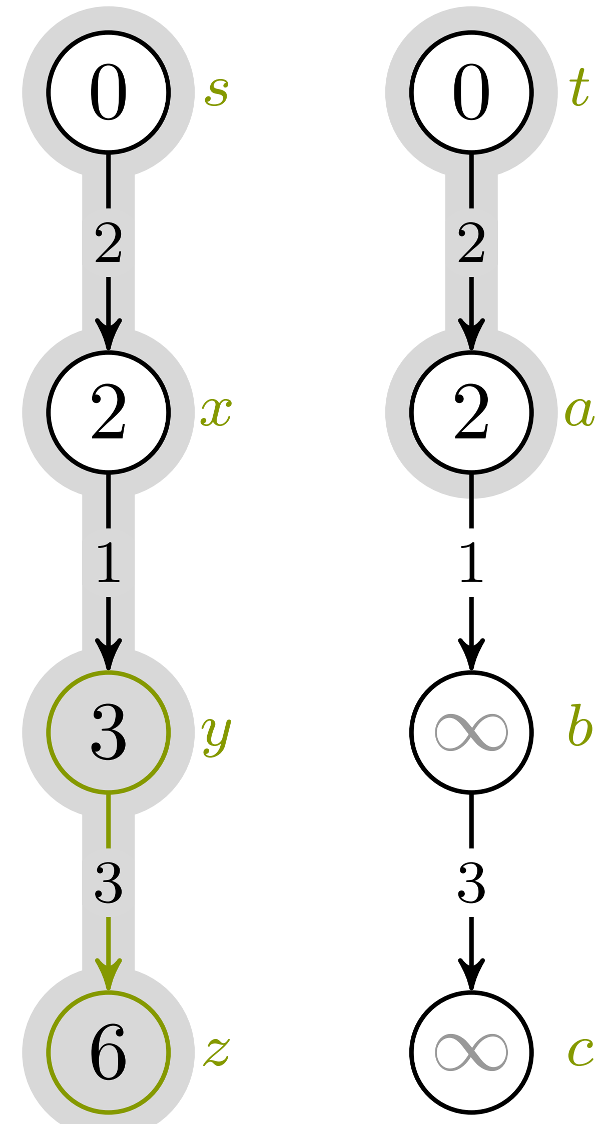
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



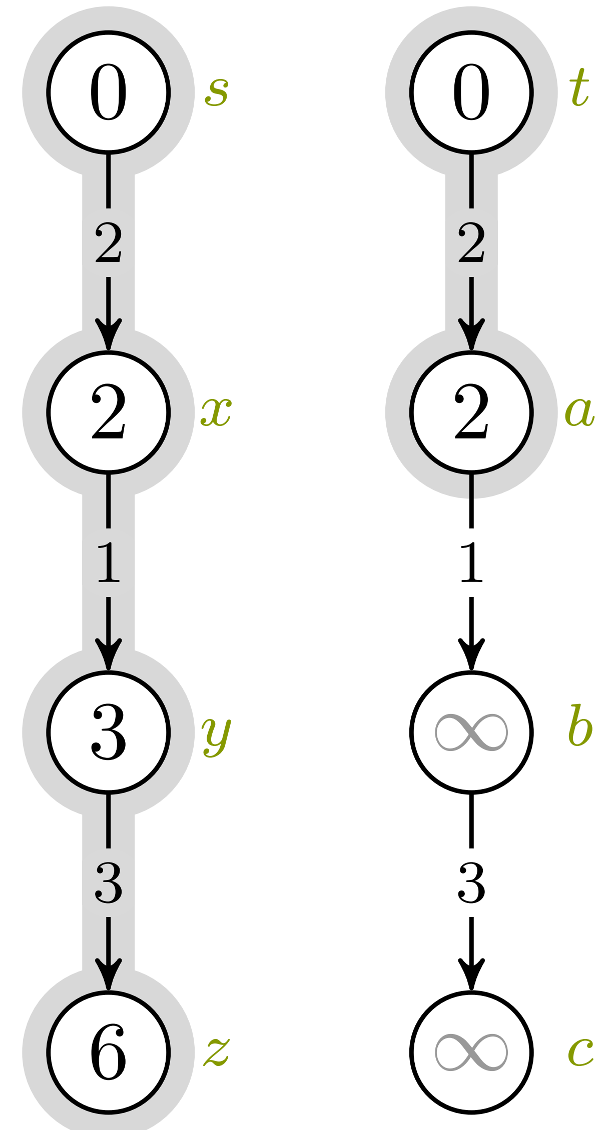
1 $\text{Relax}(s, x)$
2 $\text{Relax}(x, y)$
3 $\text{Relax}(y, z)$
4 $\text{Relax}(b, c)$
5 $\text{Relax}(a, b)$
6 $\text{Relax}(t, a)$



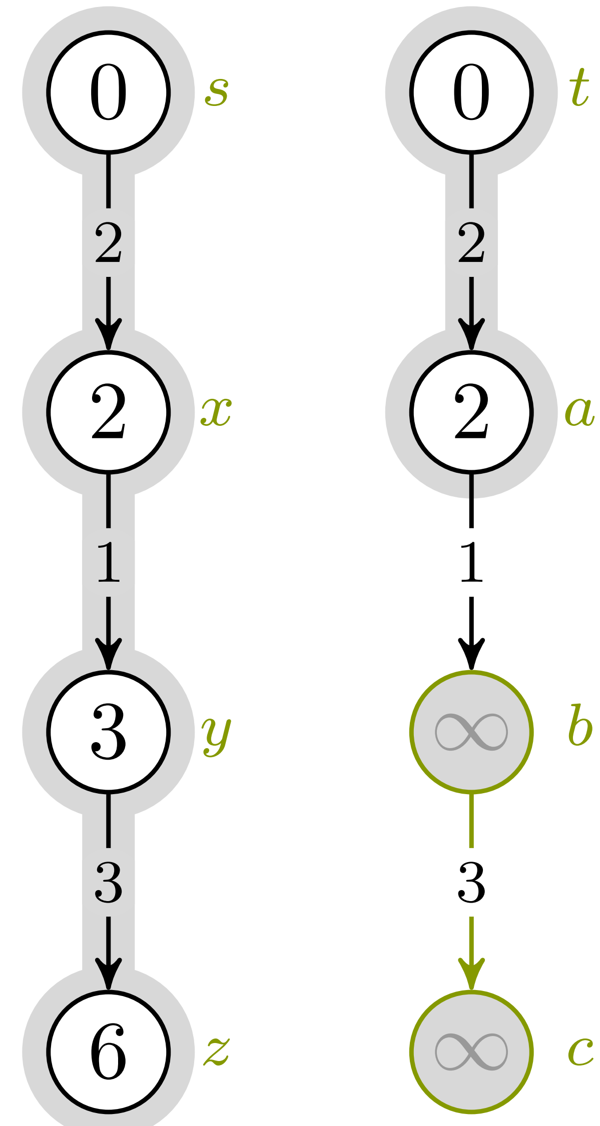
RELAX(s, x)
RELAX(x, y)
RELAX(y, z)
RELAX(b, c)
RELAX(a, b)
RELAX(t, a)



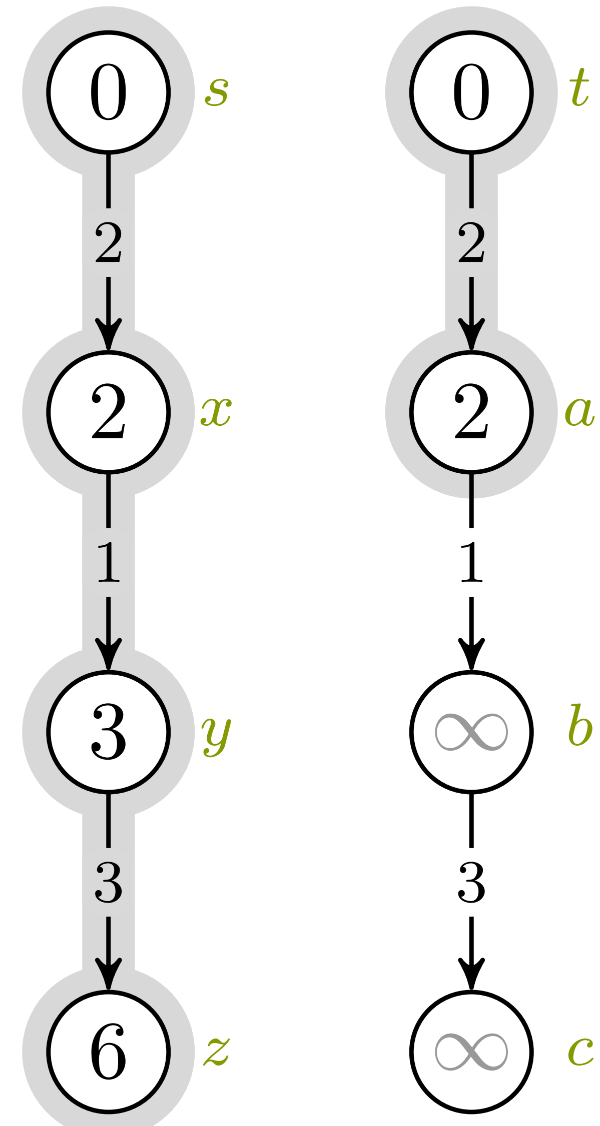
1 RELAX(s, x)
2 RELAX(x, y)
3 RELAX(y, z)
4 RELAX(b, c)
5 RELAX(a, b)
6 RELAX(t, a)



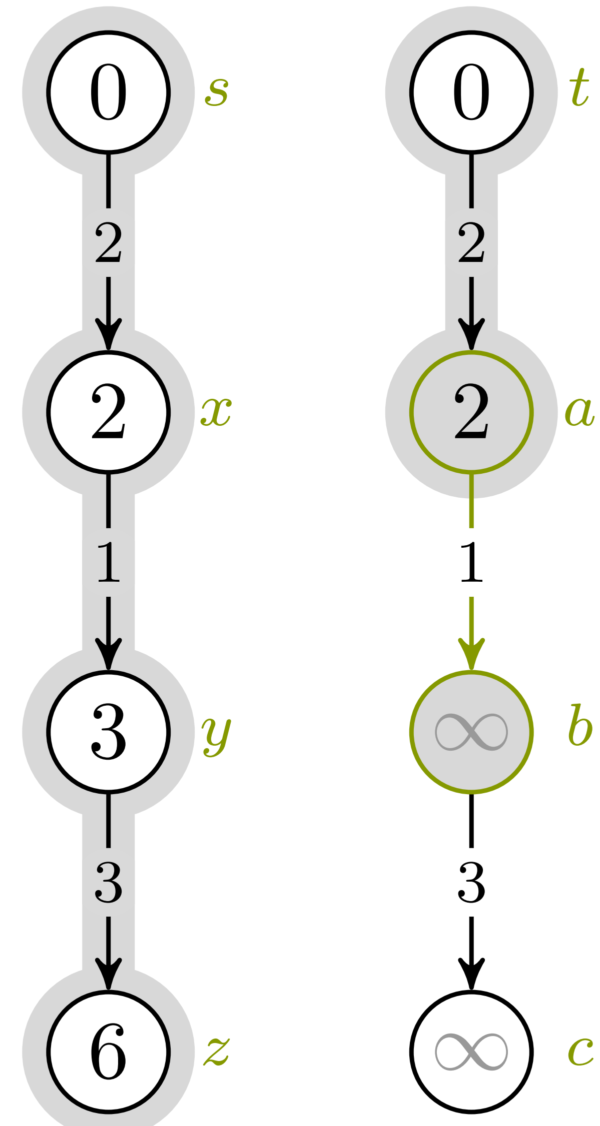
RELAX(s, x)
RELAX(x, y)
RELAX(y, z)
RELAX(b, c)
RELAX(a, b)
RELAX(t, a)



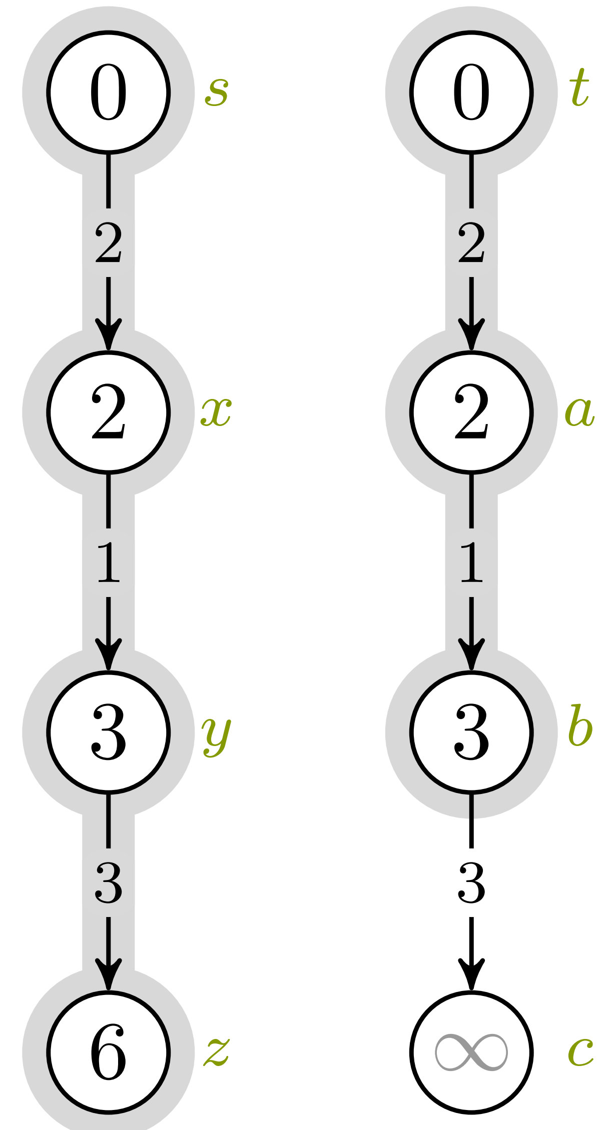
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



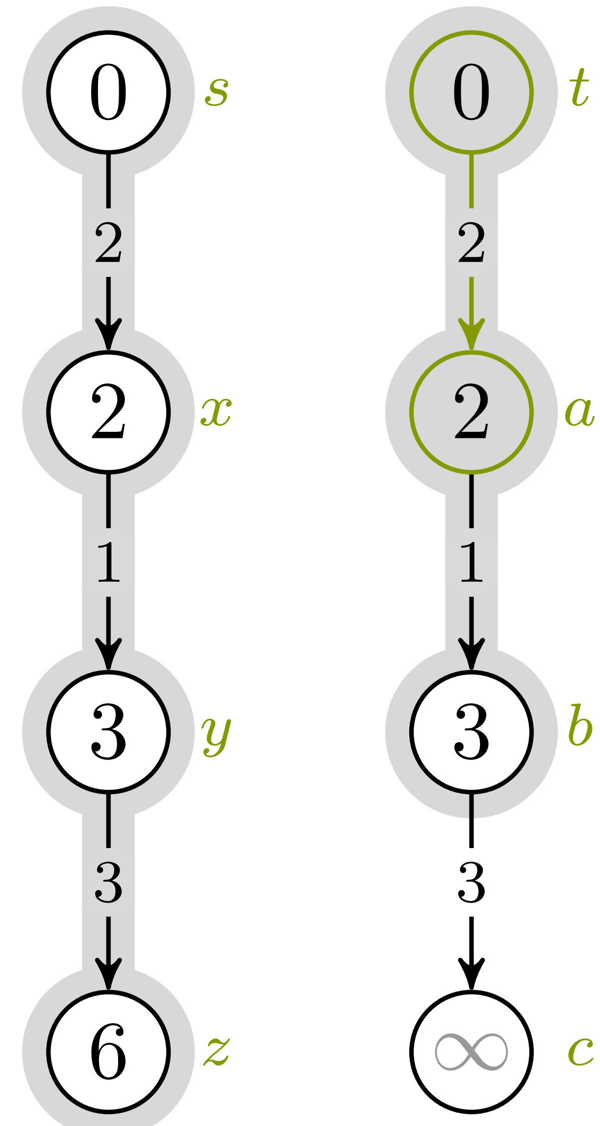
$\textsc{Relax}(s, x)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(b, c)$
$\textsc{Relax}(a, b)$
$\textsc{Relax}(t, a)$



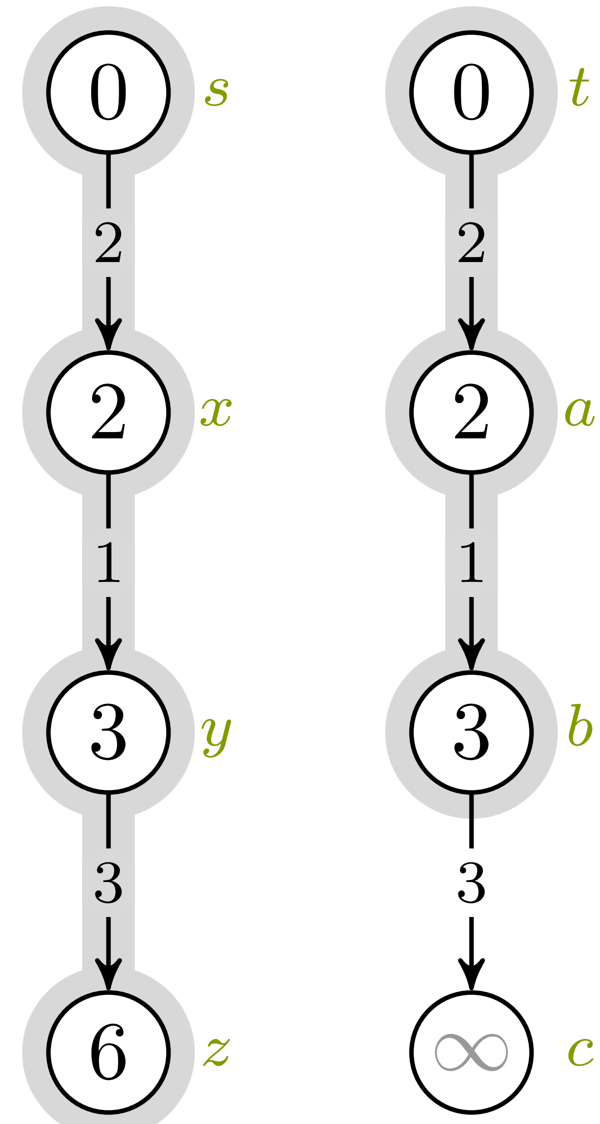
1 Relax(s, x)
2 Relax(x, y)
3 Relax(y, z)
4 Relax(b, c)
5 Relax(a, b)
6 Relax(t, a)



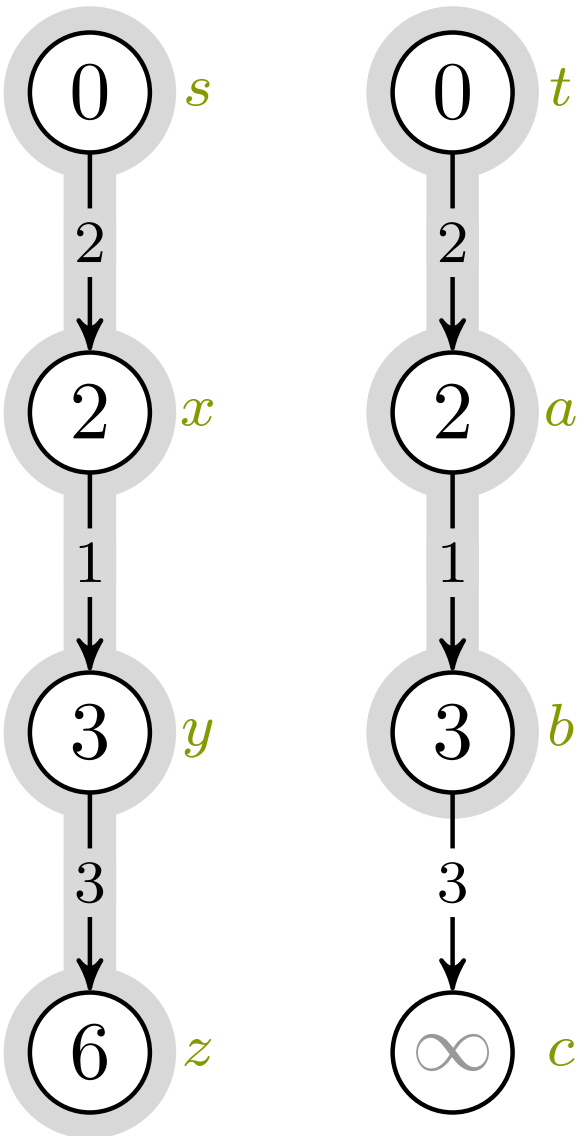
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)



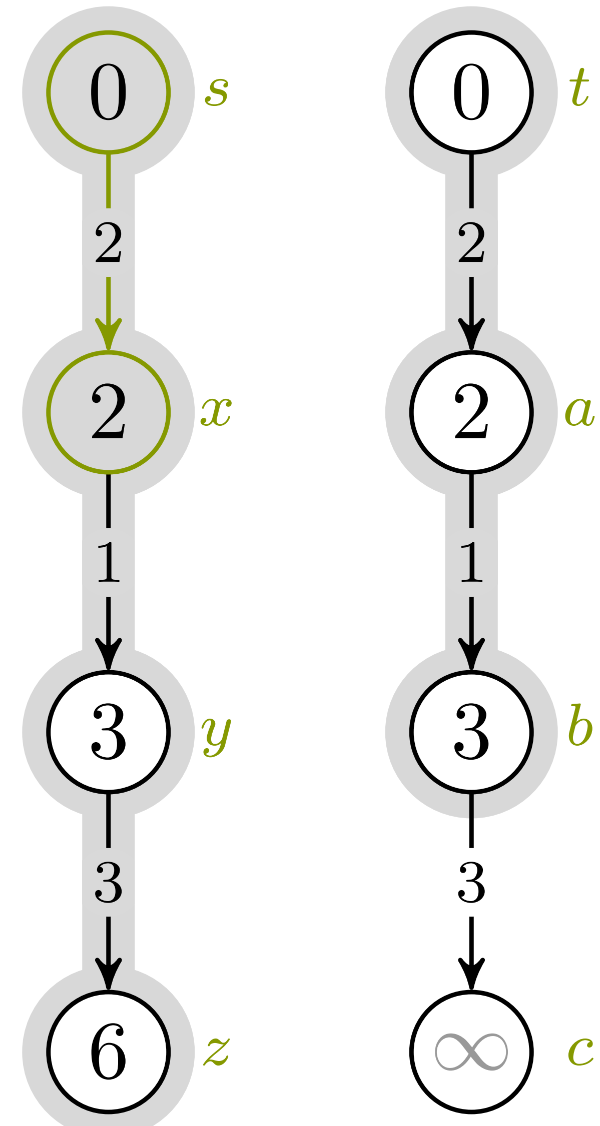
1 Relax(s, x)
2 Relax(x, y)
3 Relax(y, z)
4 Relax(b, c)
5 Relax(a, b)
6 Relax(t, a)



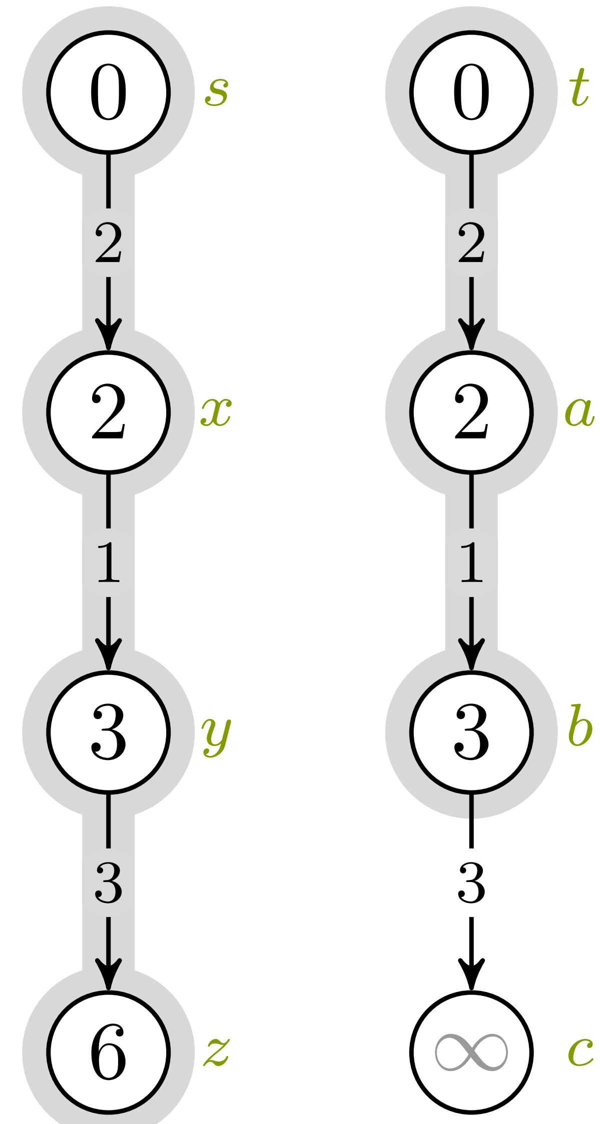
1 RELAX(s, x)
2 RELAX(x, y)
3 RELAX(y, z)
4 RELAX(b, c)
5 RELAX(a, b)
6 RELAX(t, a)



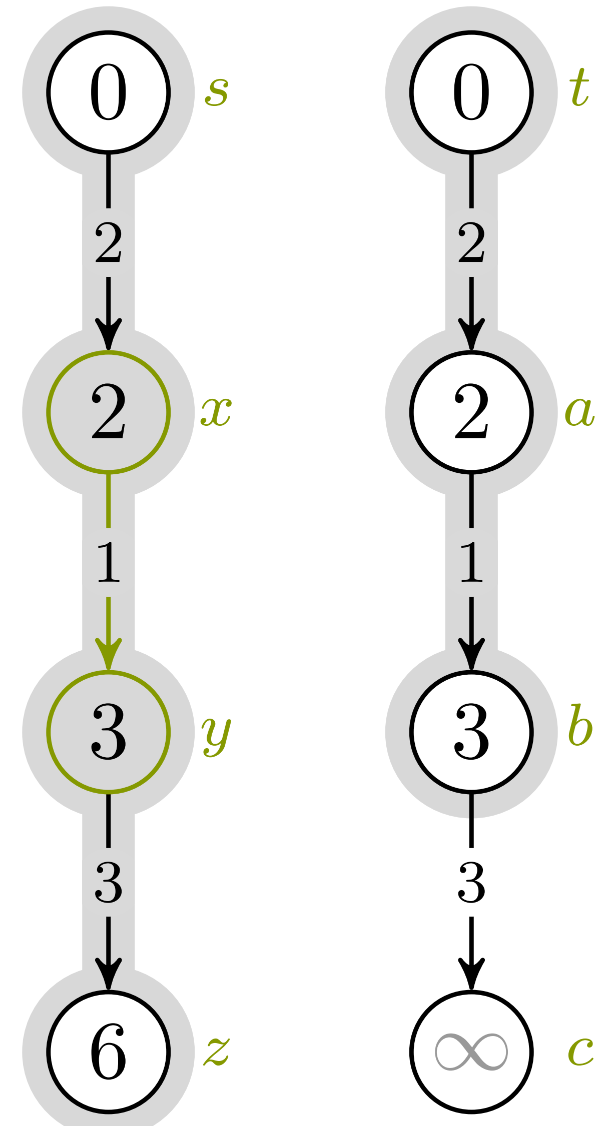
```
RELAX(s, x)
RELAX(x, y)
RELAX(y, z)
RELAX(b, c)
RELAX(a, b)
RELAX(t, a)
```



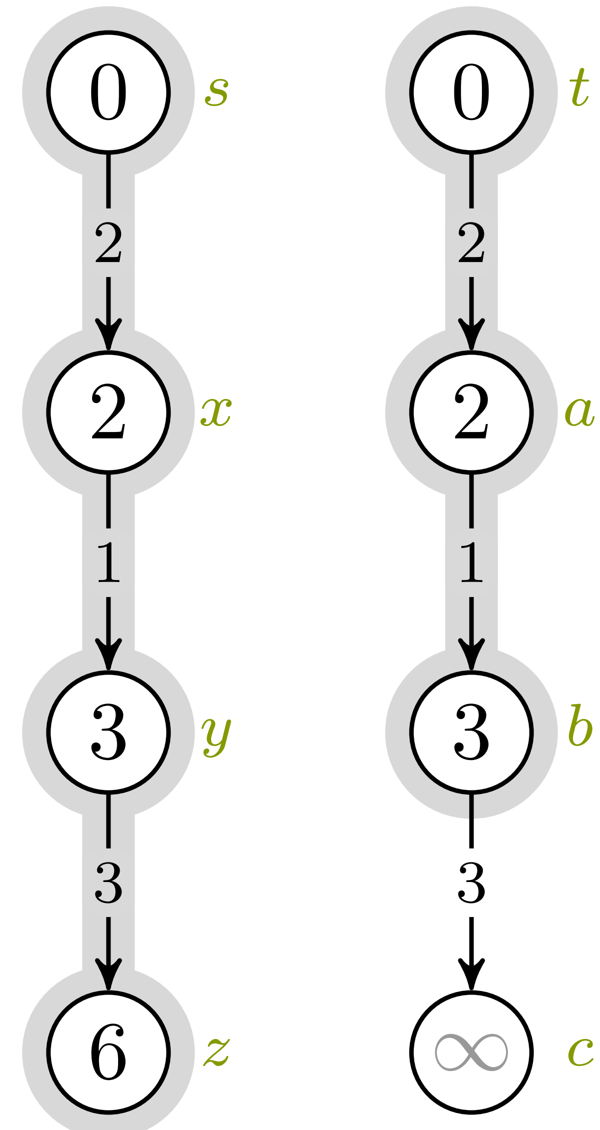
1. $\textsc{Relax}(s, x)$
2. $\textsc{Relax}(x, y)$
3. $\textsc{Relax}(y, z)$
4. $\textsc{Relax}(b, c)$
5. $\textsc{Relax}(a, b)$
6. $\textsc{Relax}(t, a)$



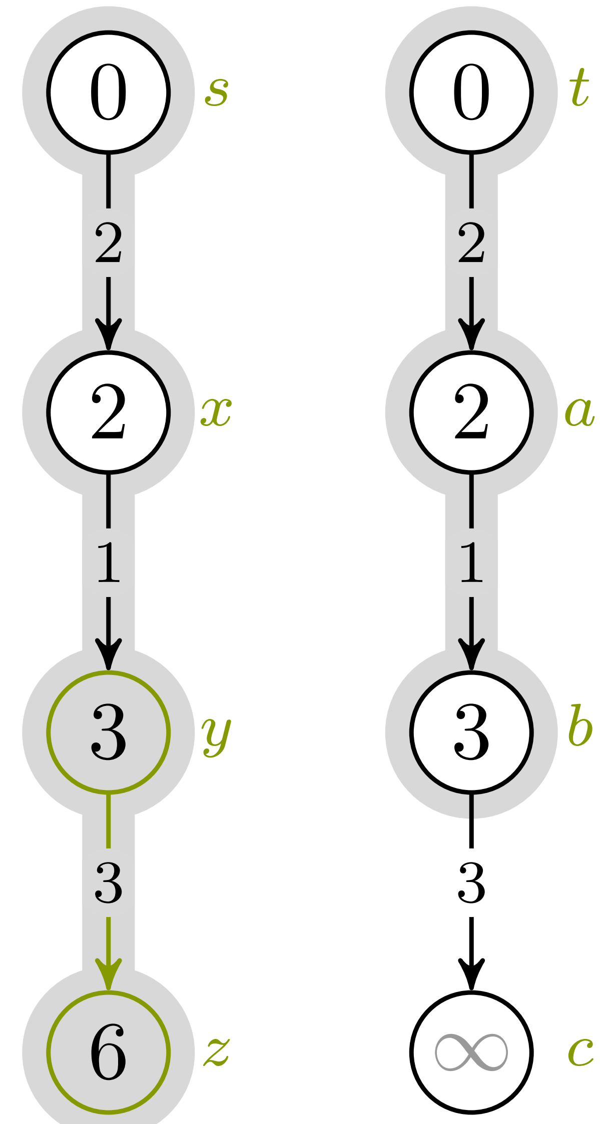
```
Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)
```



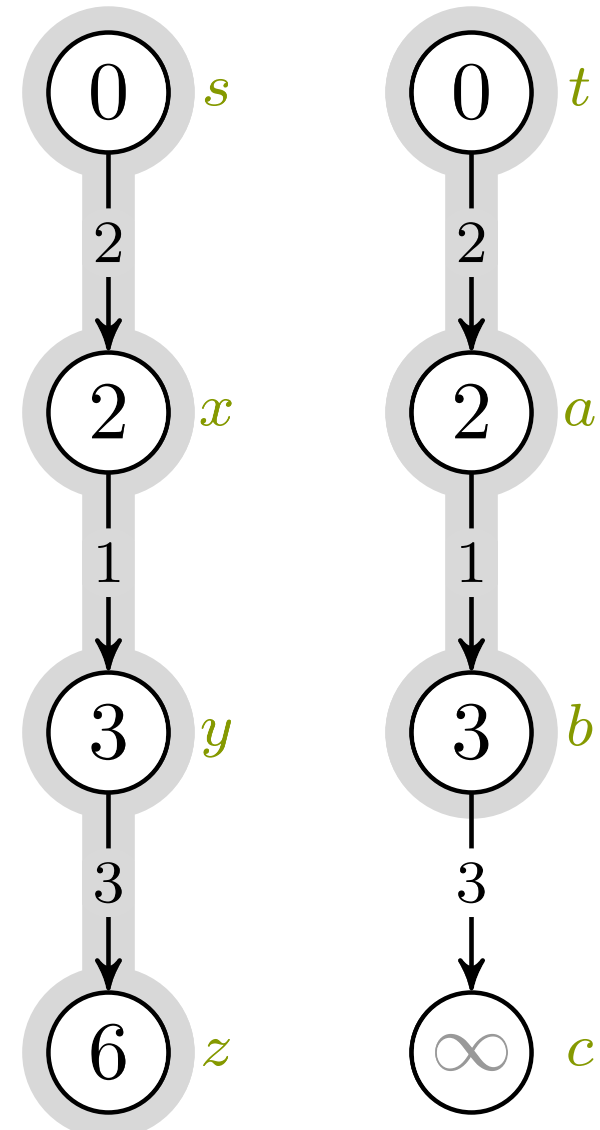
1 Relax(s, x)
2 Relax(x, y)
3 Relax(y, z)
4 Relax(b, c)
5 Relax(a, b)
6 Relax(t, a)



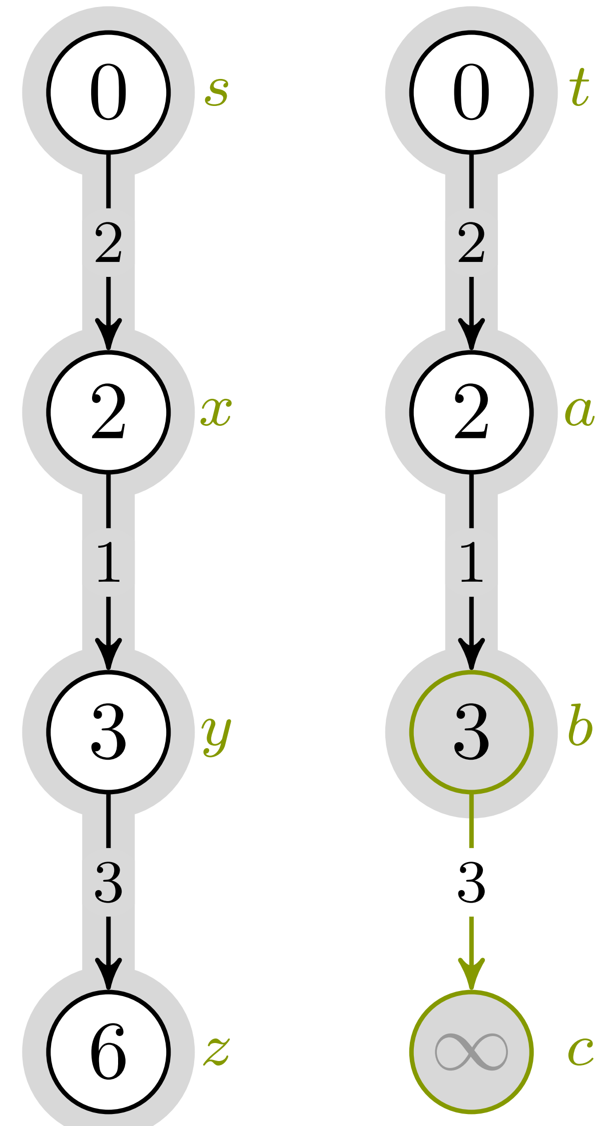
1 Relax(s, x)
2 Relax(x, y)
3 Relax(y, z)
4 Relax(b, c)
5 Relax(a, b)
6 Relax(t, a)



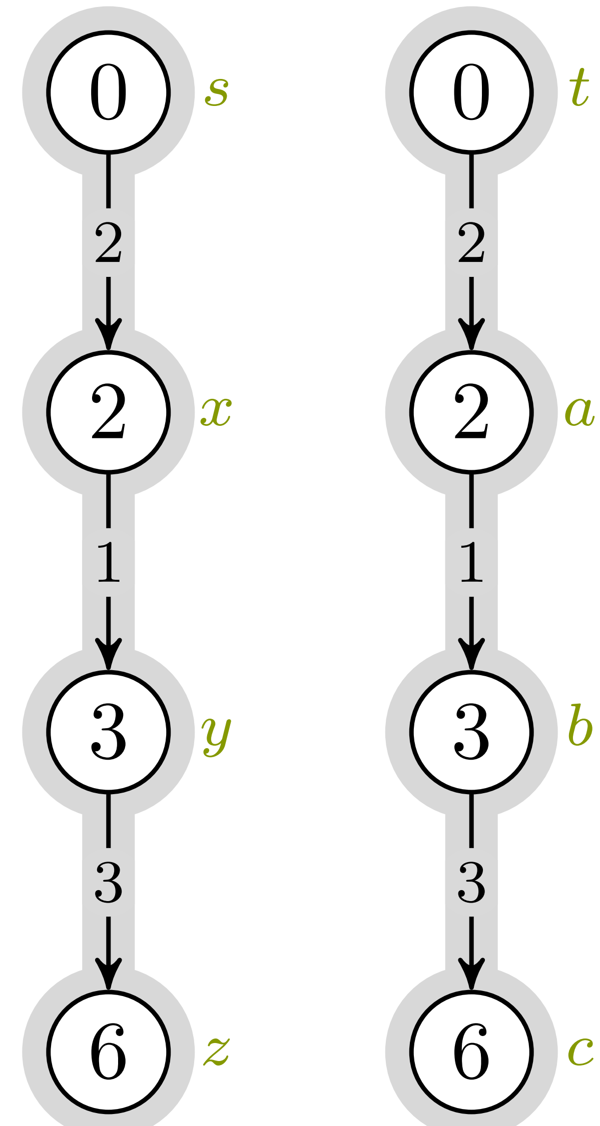
1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(y, z)$
4 $\textsc{Relax}(b, c)$
5 $\textsc{Relax}(a, b)$
6 $\textsc{Relax}(t, a)$



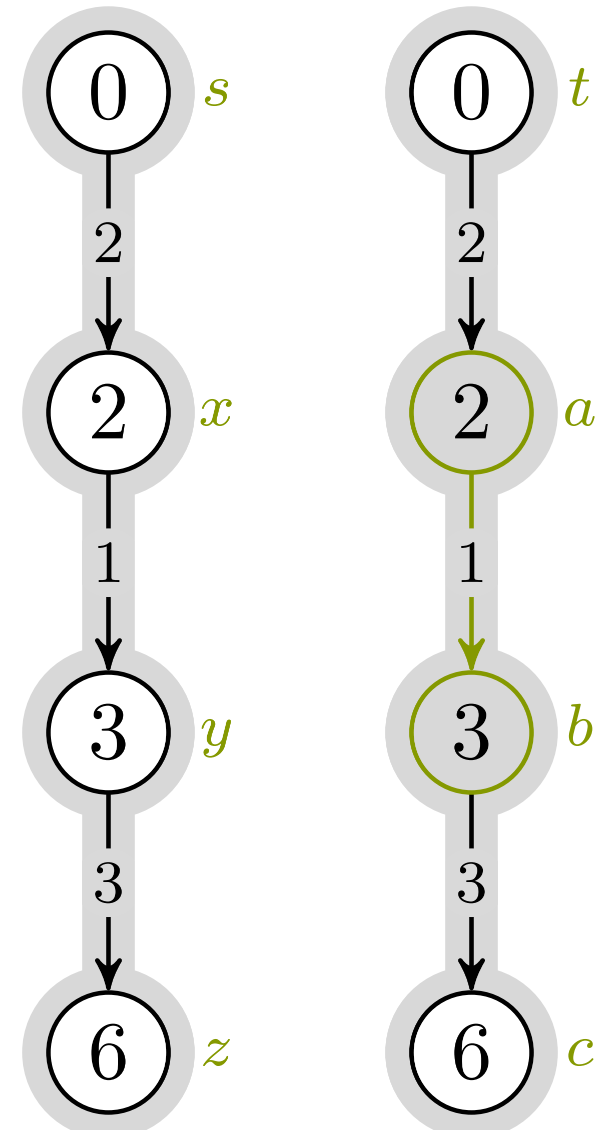
RELAX(s, x)
RELAX(x, y)
RELAX(y, z)
RELAX(b, c)
RELAX(a, b)
RELAX(t, a)



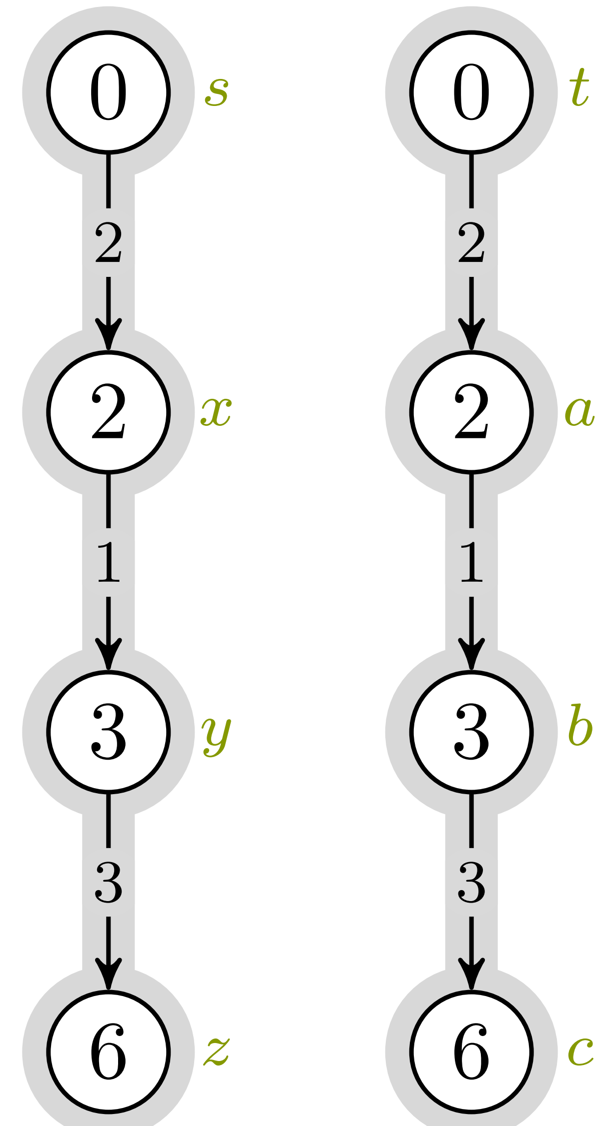
$\text{Relax}(s, x)$
$\text{Relax}(x, y)$
$\text{Relax}(y, z)$
$\text{Relax}(b, c)$
$\text{Relax}(a, b)$
$\text{Relax}(t, a)$



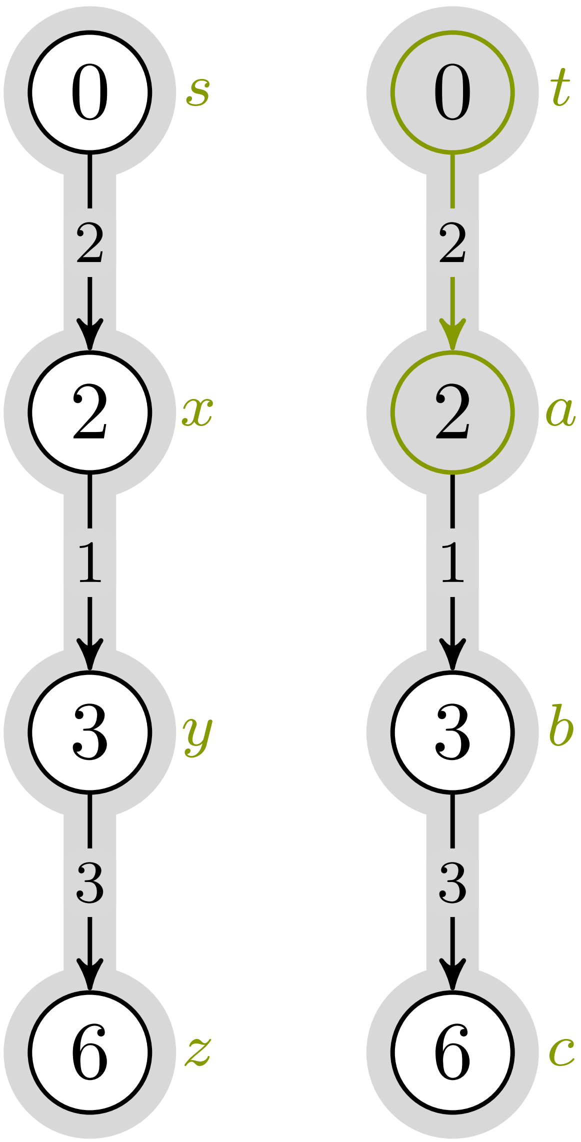
1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(y, z)$
4 $\textsc{Relax}(b, c)$
5 $\textsc{Relax}(a, b)$
6 $\textsc{Relax}(t, a)$



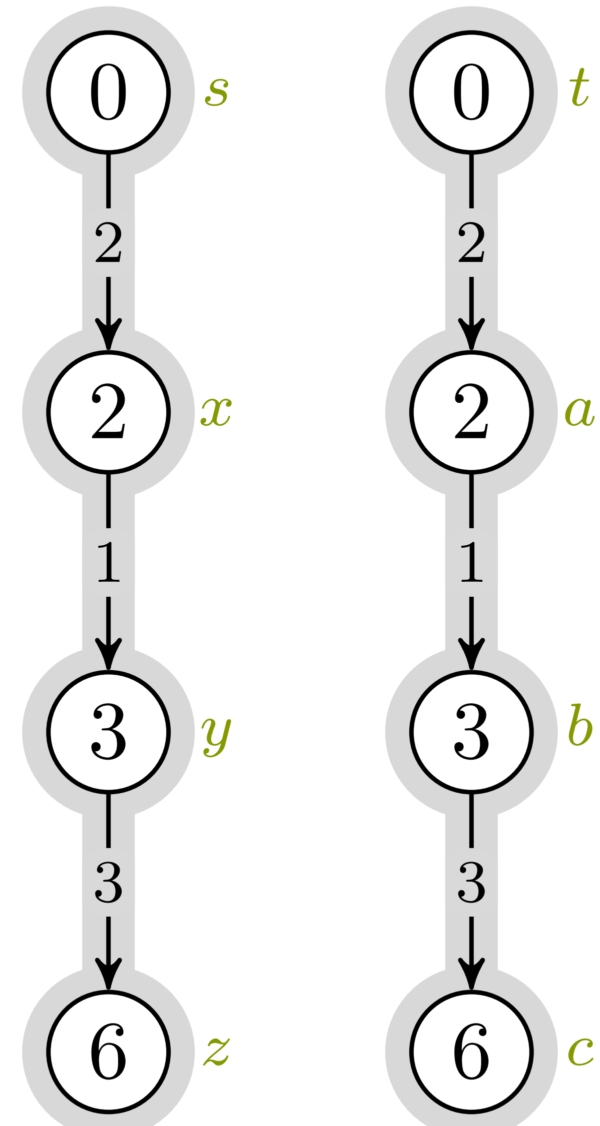
RELAX(s, x)
RELAX(x, y)
RELAX(y, z)
RELAX(b, c)
RELAX(a, b)
RELAX(t, a)



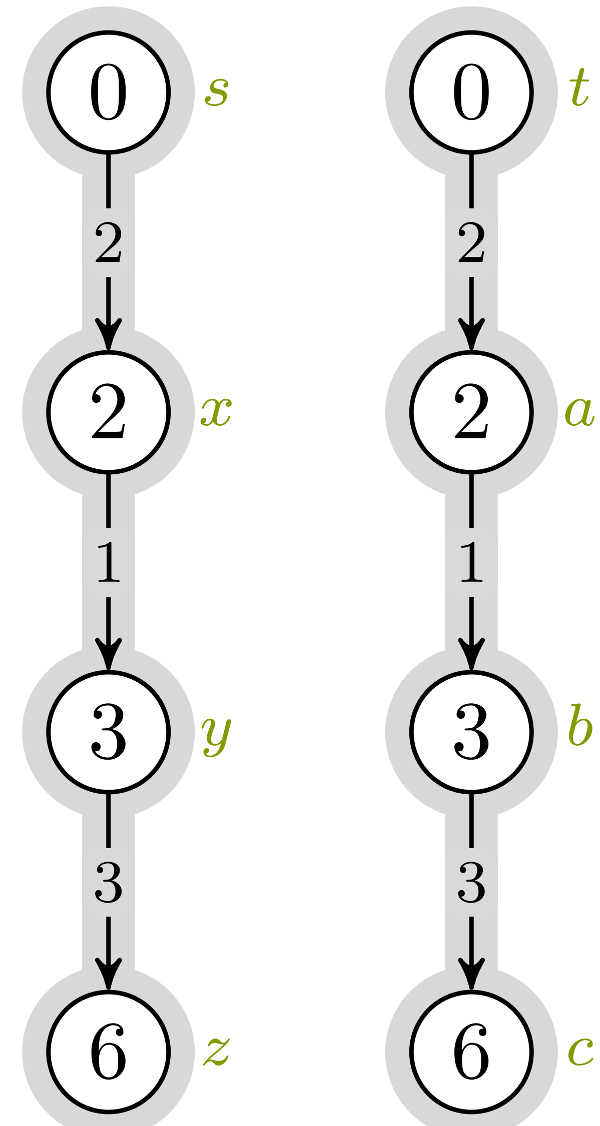
1 $\textsc{Relax}(s, x)$
2 $\textsc{Relax}(x, y)$
3 $\textsc{Relax}(y, z)$
4 $\textsc{Relax}(b, c)$
5 $\textsc{Relax}(a, b)$
6 $\textsc{Relax}(t, a)$



Relax(s, x)
Relax(x, y)
Relax(y, z)
Relax(b, c)
Relax(a, b)
Relax(t, a)

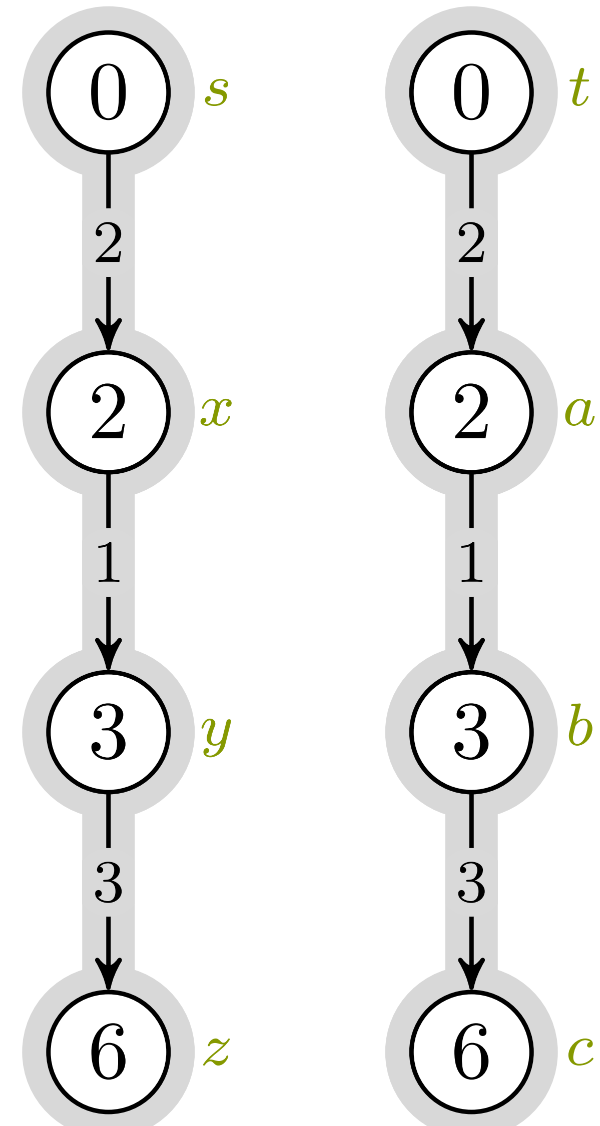


$\textsc{Relax}(s, x)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(b, c)$
$\textsc{Relax}(a, b)$
$\textsc{Relax}(t, a)$



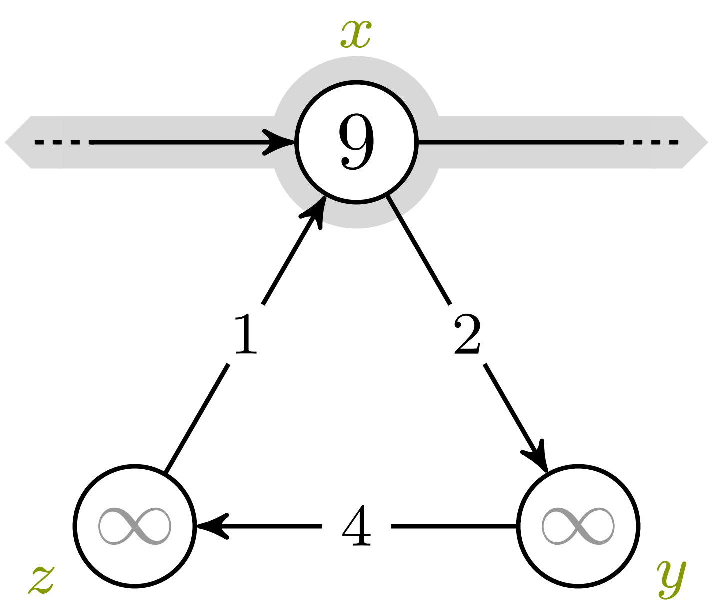
Tre pass: Garantert ferdig

RELAX(s, x)
RELAX(x, y)
RELAX(y, z)
RELAX(b, c)
RELAX(a, b)
RELAX(t, a)



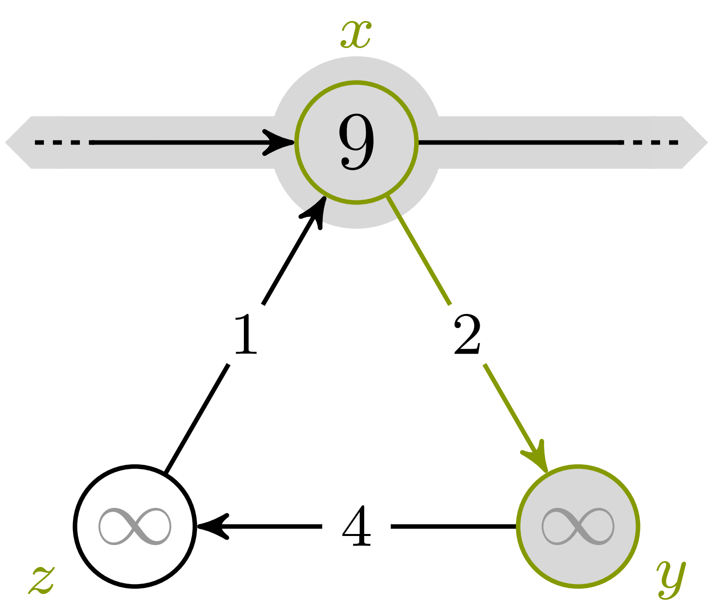
Hvorfor akkurat tre?

1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$

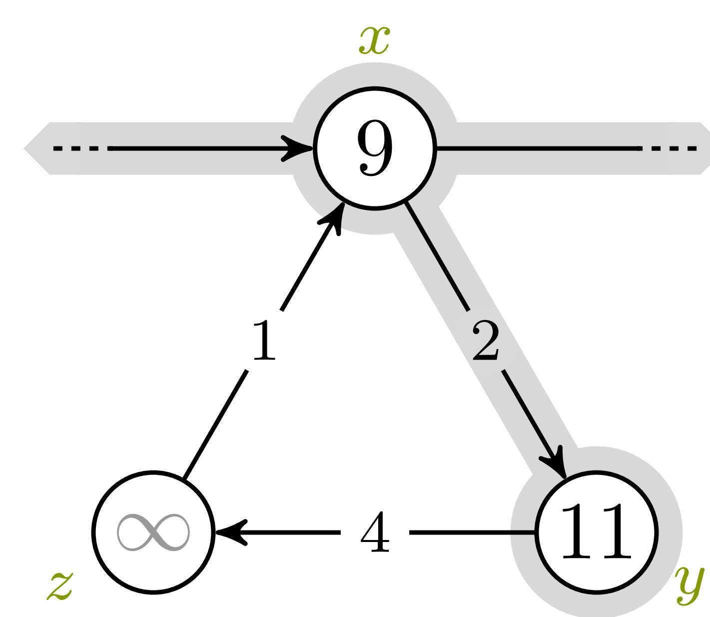


Positiv sykel

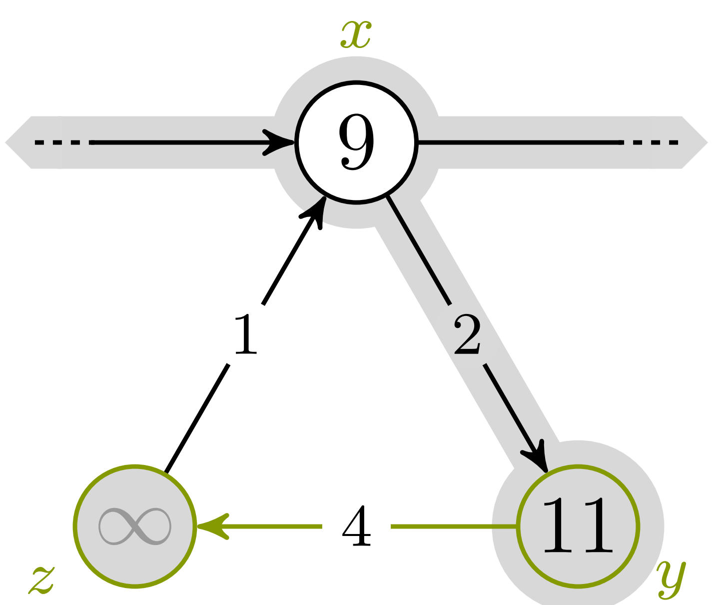
1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$



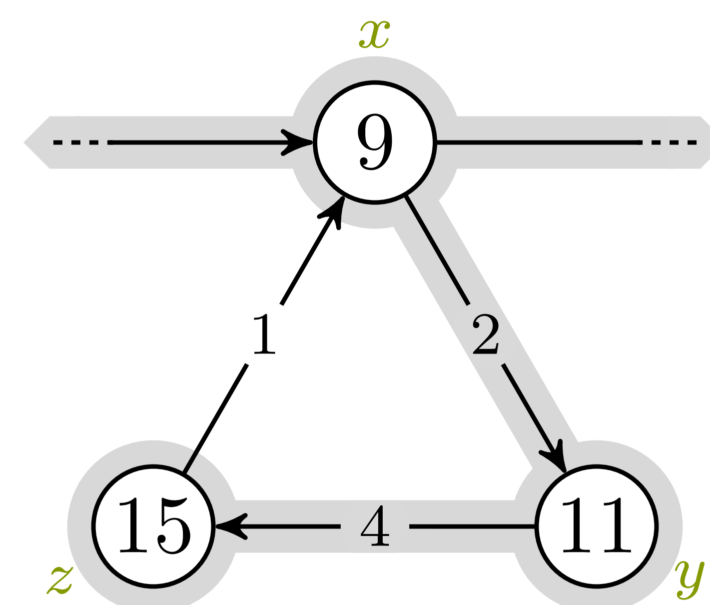
1 RELAX(x, y)
2 RELAX(y, z)
3 RELAX(z, x)



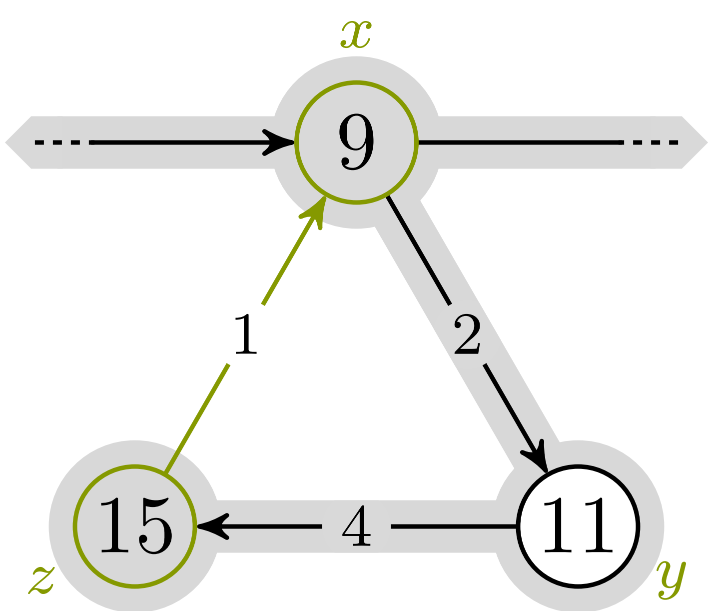
1 Relax(x, y)
2 Relax(y, z)
3 Relax(z, x)



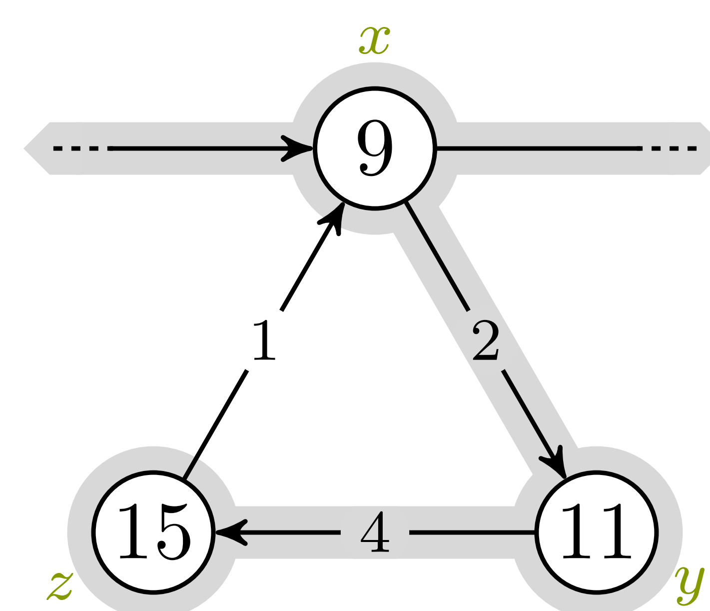
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



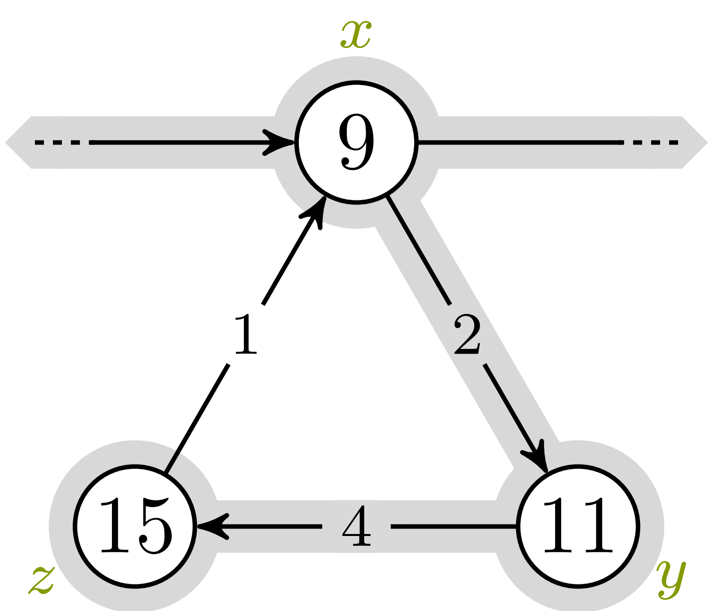
1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$



1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$

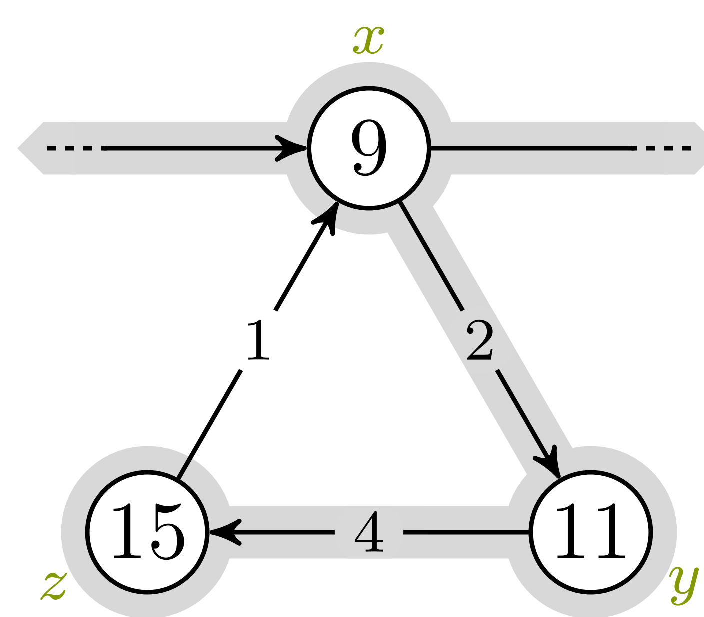


1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$



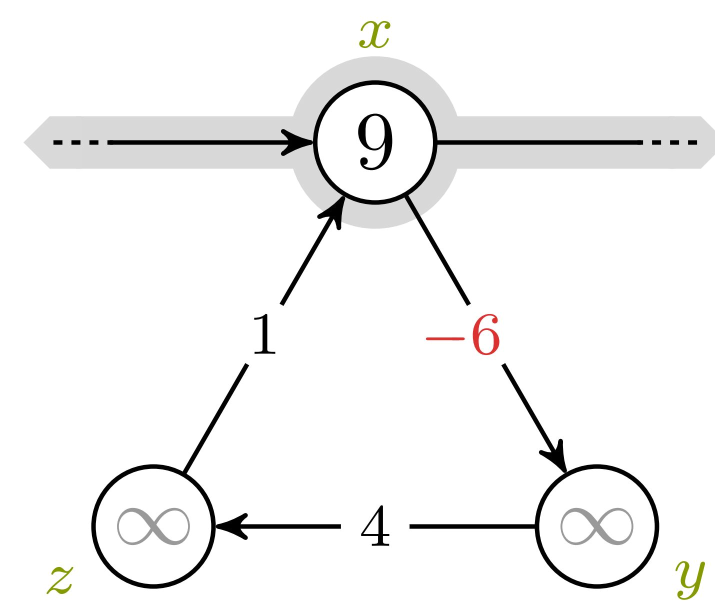
Unødvendig ekstrakostnad

1 $\text{Relax}(x, y)$
2 $\text{Relax}(y, z)$
3 $\text{Relax}(z, x)$



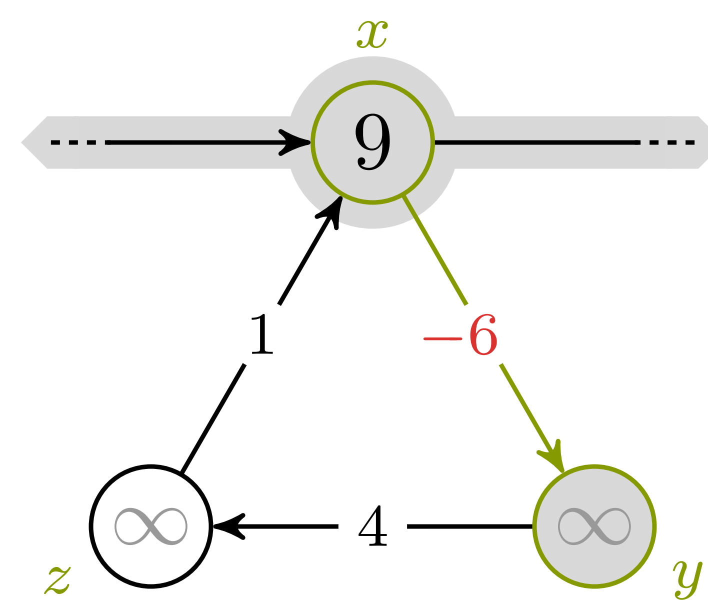
Vil ikke bli del av kortest vei

$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(z, x)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(z, x)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(z, x)$

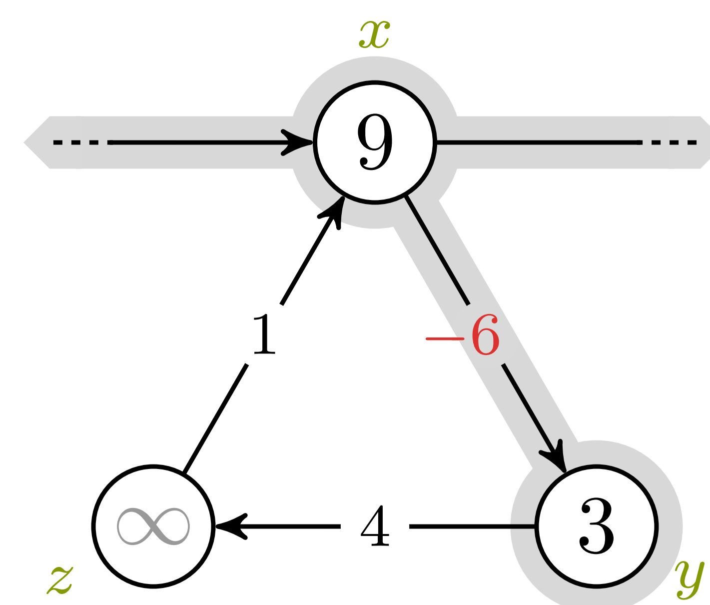


Negativ sykel

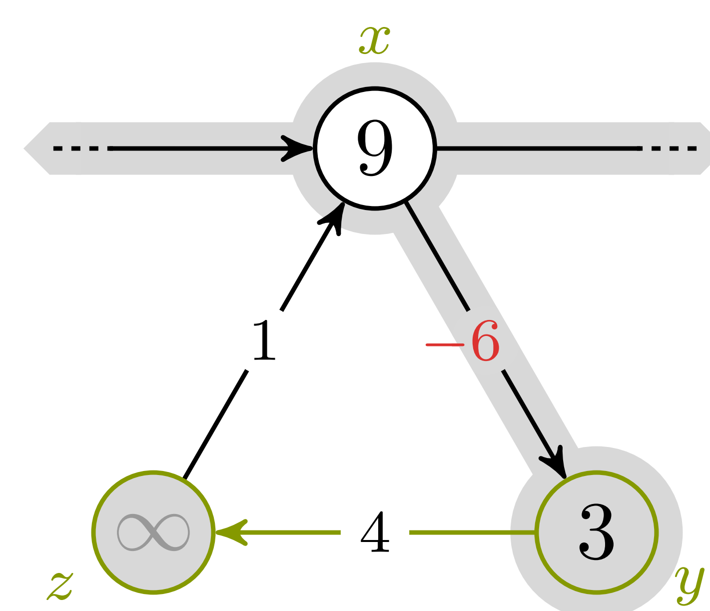
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



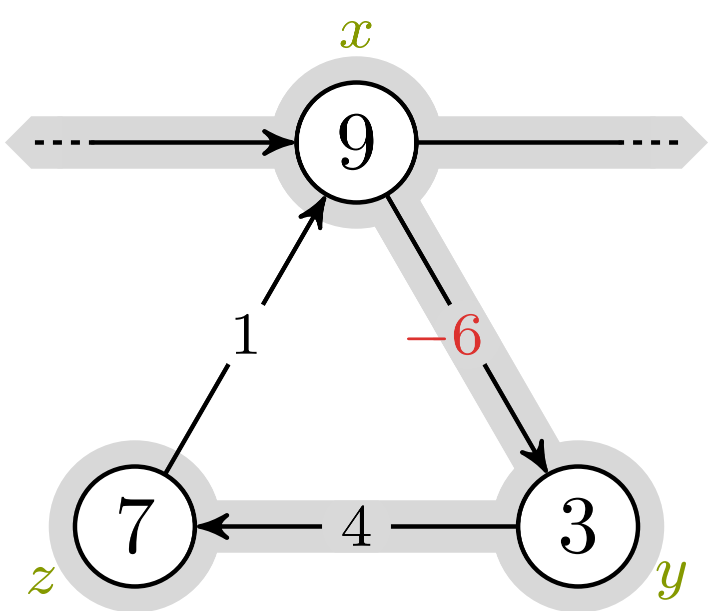
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



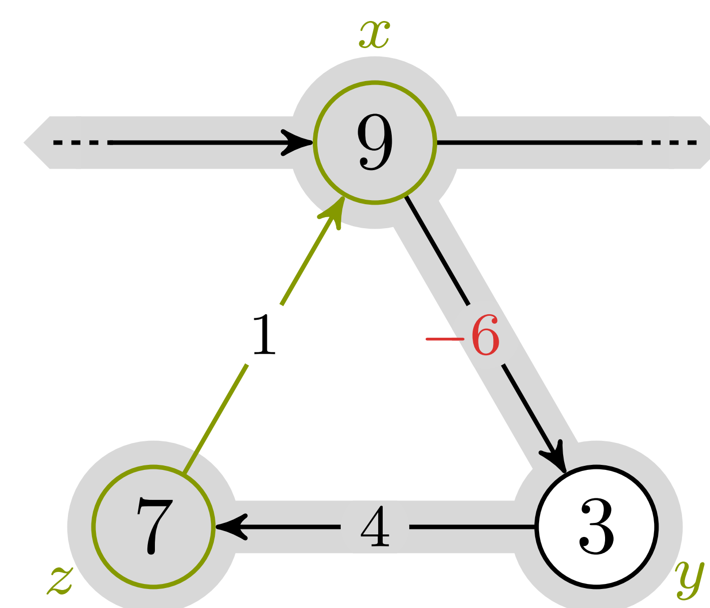
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(z, x)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(z, x)$
$\textsc{Relax}(x, y)$
$\textsc{Relax}(y, z)$
$\textsc{Relax}(z, x)$



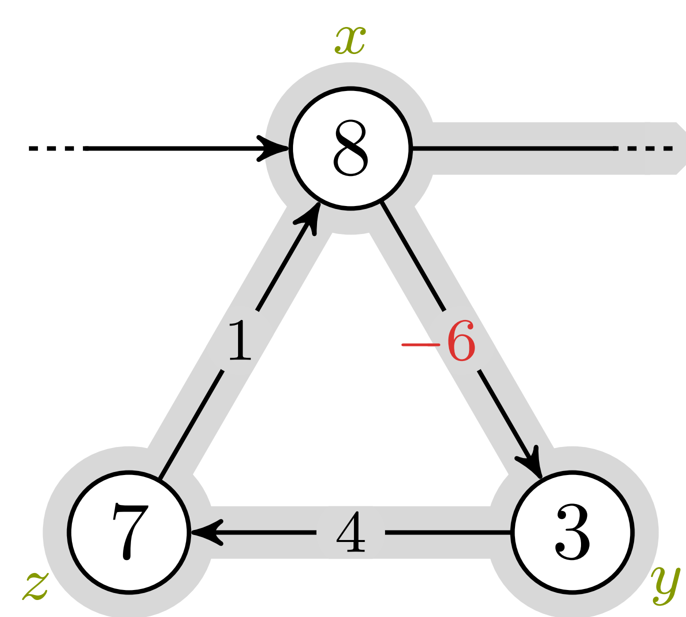
1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$
4 $\textsc{Relax}(x, y)$
5 $\textsc{Relax}(y, z)$
6 $\textsc{Relax}(z, x)$
7 $\textsc{Relax}(x, y)$
8 $\textsc{Relax}(y, z)$
9 $\textsc{Relax}(z, x)$



```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```

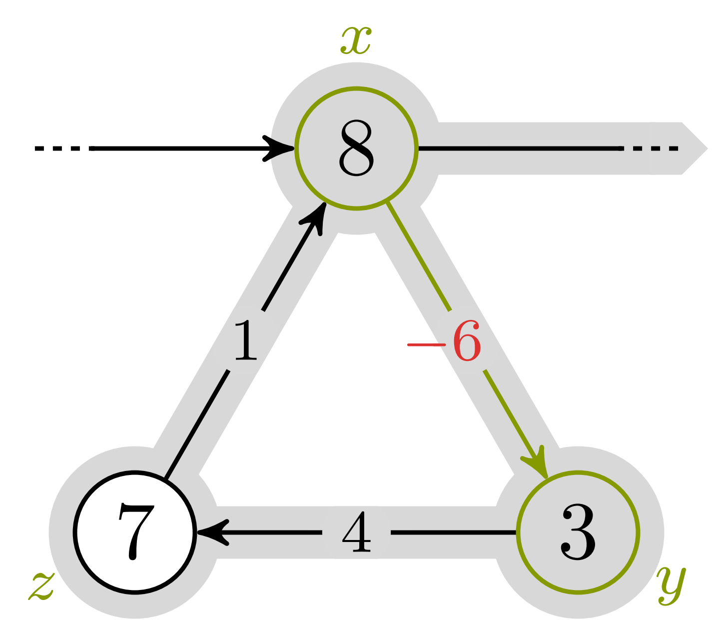


1 Relax(x, y)
2 Relax(y, z)
3 Relax(z, x)
4 Relax(x, y)
5 Relax(y, z)
6 Relax(z, x)
7 Relax(x, y)
8 Relax(y, z)
9 Relax(z, x)

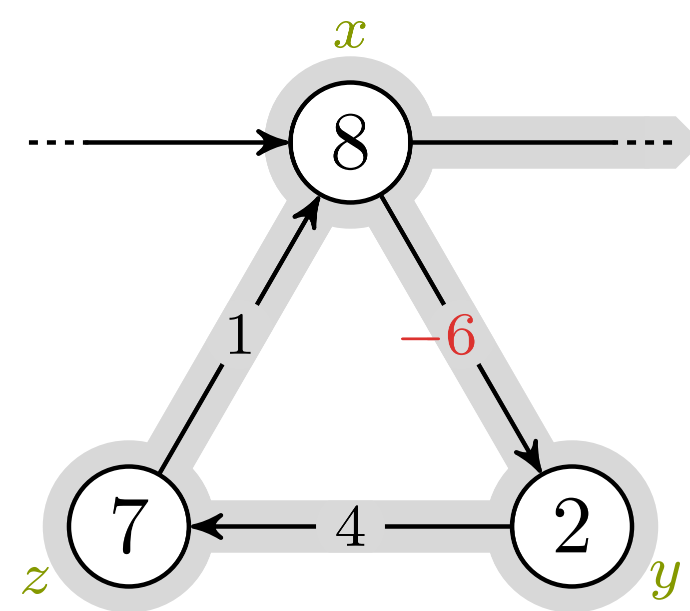


(Impl.: Har kun ett $x.\pi$-felt)

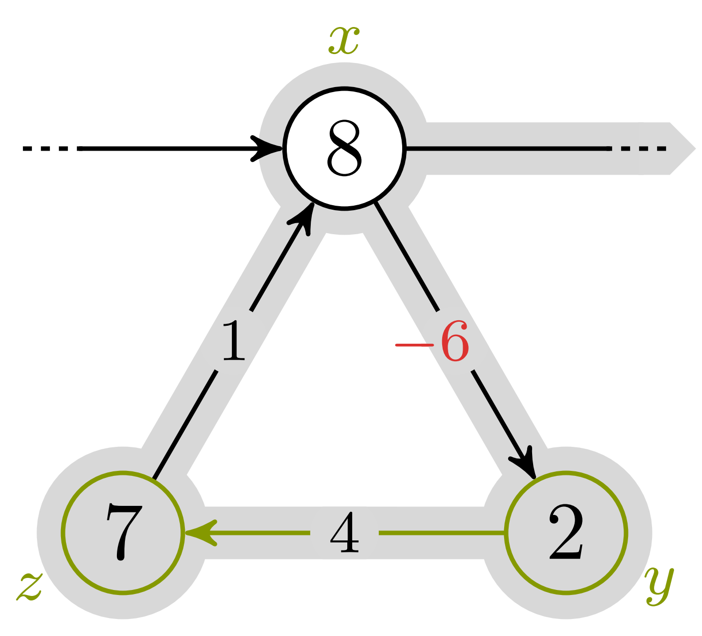
1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$
4 $\textsc{Relax}(x, y)$
5 $\textsc{Relax}(y, z)$
6 $\textsc{Relax}(z, x)$
7 $\textsc{Relax}(x, y)$
8 $\textsc{Relax}(y, z)$
9 $\textsc{Relax}(z, x)$



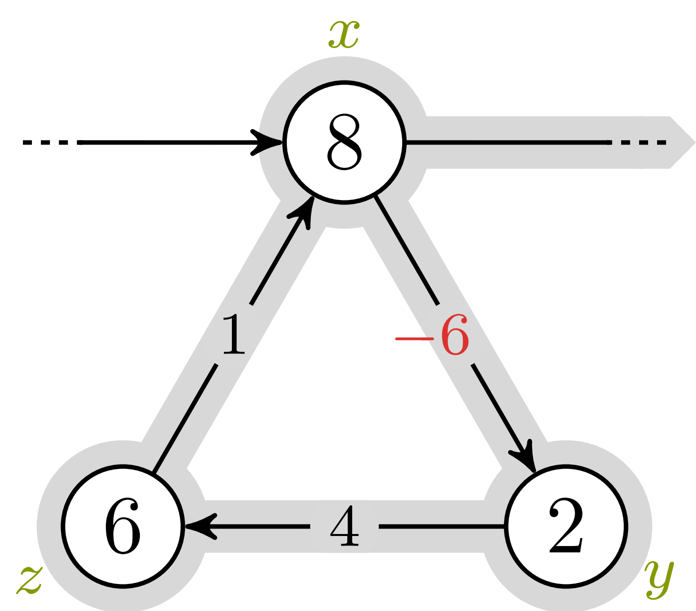
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



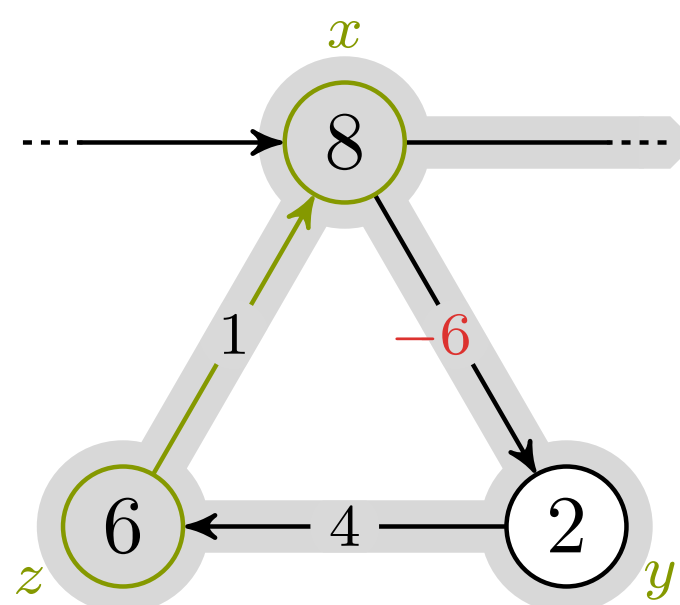
1 Relax(x, y)
2 Relax(y, z)
3 Relax(z, x)
4 Relax(x, y)
5 Relax(y, z)
6 Relax(z, x)
7 Relax(x, y)
8 Relax(y, z)
9 Relax(z, x)



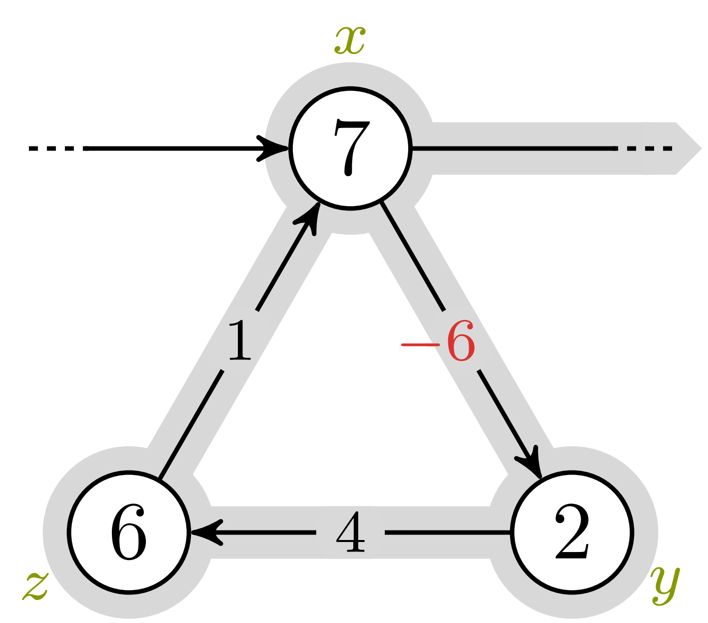
1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$
4 $\textsc{Relax}(x, y)$
5 $\textsc{Relax}(y, z)$
6 $\textsc{Relax}(z, x)$
7 $\textsc{Relax}(x, y)$
8 $\textsc{Relax}(y, z)$
9 $\textsc{Relax}(z, x)$



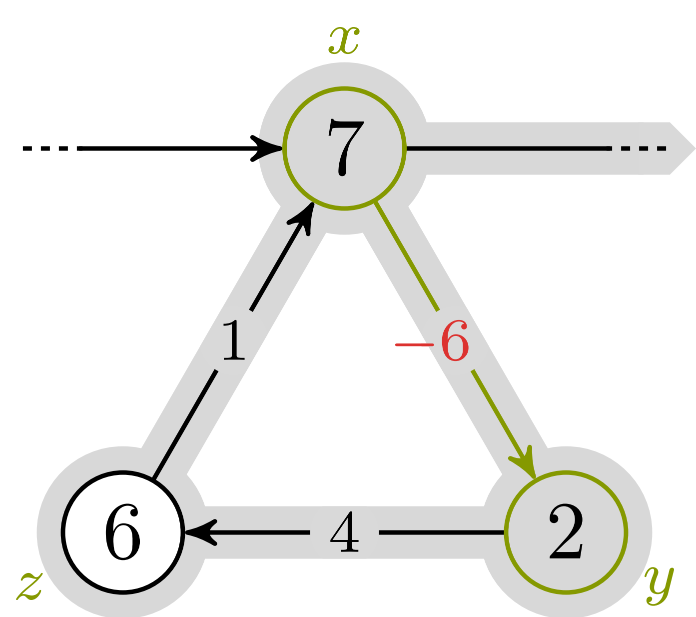
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



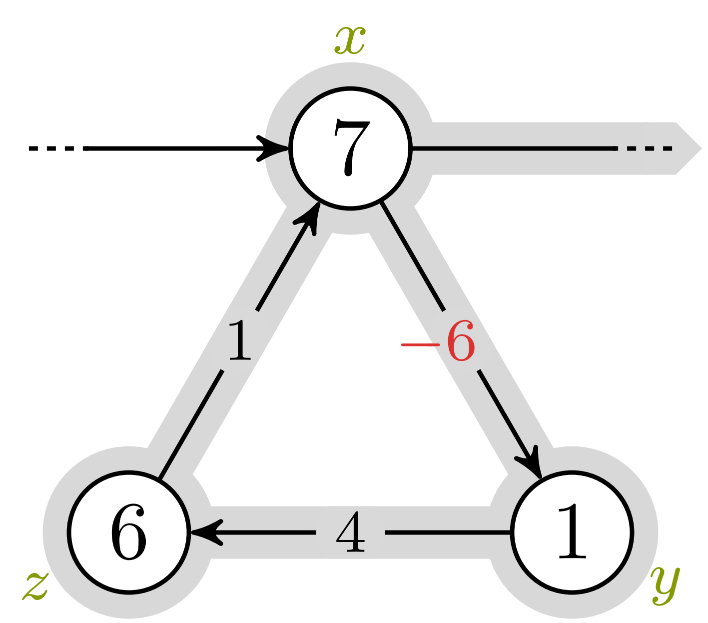
1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$
4 $\textsc{Relax}(x, y)$
5 $\textsc{Relax}(y, z)$
6 $\textsc{Relax}(z, x)$
7 $\textsc{Relax}(x, y)$
8 $\textsc{Relax}(y, z)$
9 $\textsc{Relax}(z, x)$



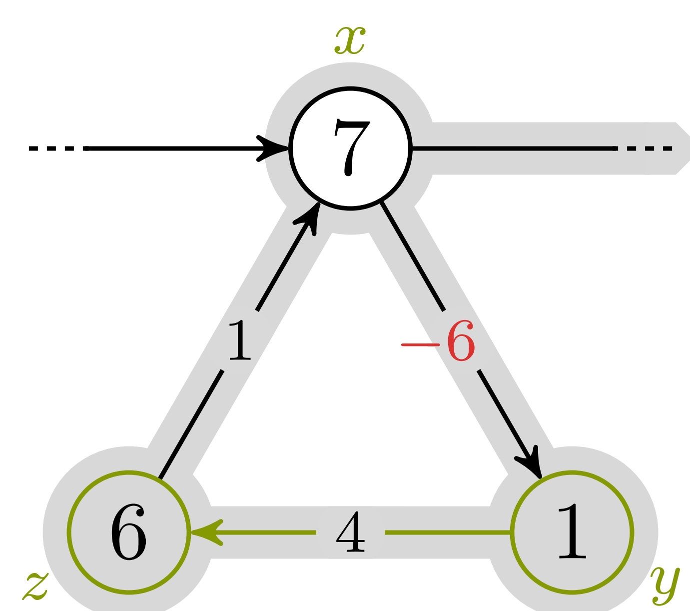
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



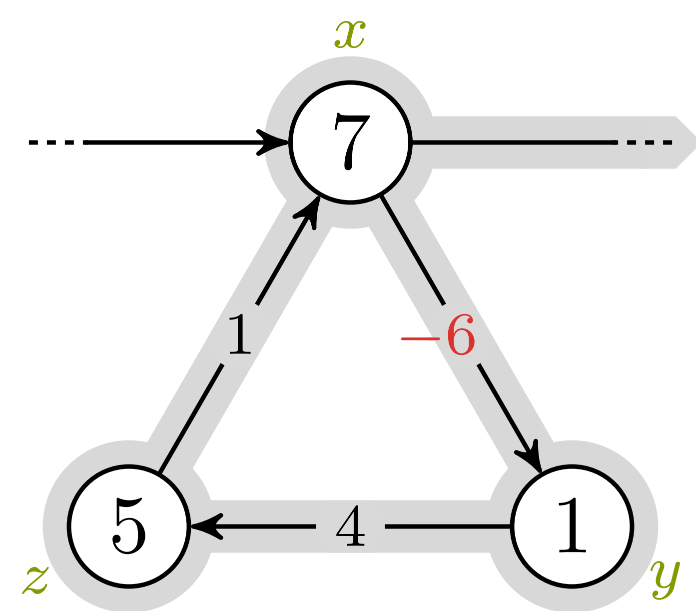
1 $\textsc{Relax}(x, y)$
2 $\textsc{Relax}(y, z)$
3 $\textsc{Relax}(z, x)$
4 $\textsc{Relax}(x, y)$
5 $\textsc{Relax}(y, z)$
6 $\textsc{Relax}(z, x)$
7 $\textsc{Relax}(x, y)$
8 $\textsc{Relax}(y, z)$
9 $\textsc{Relax}(z, x)$



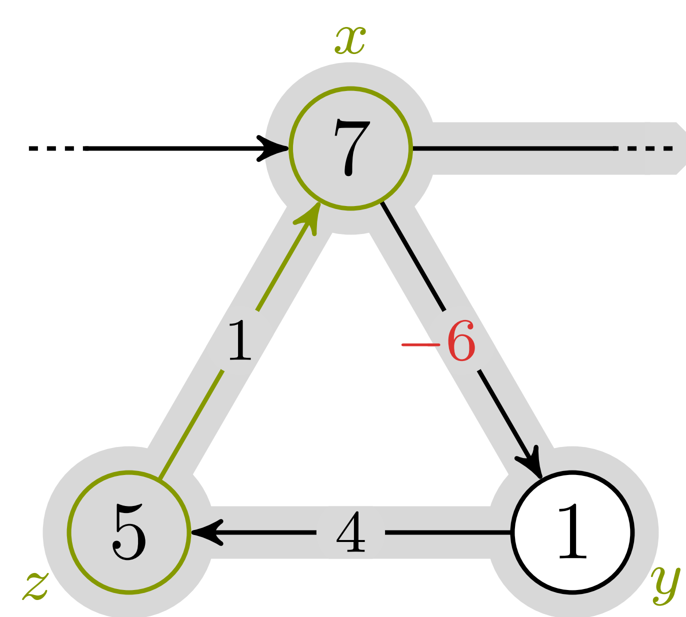
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



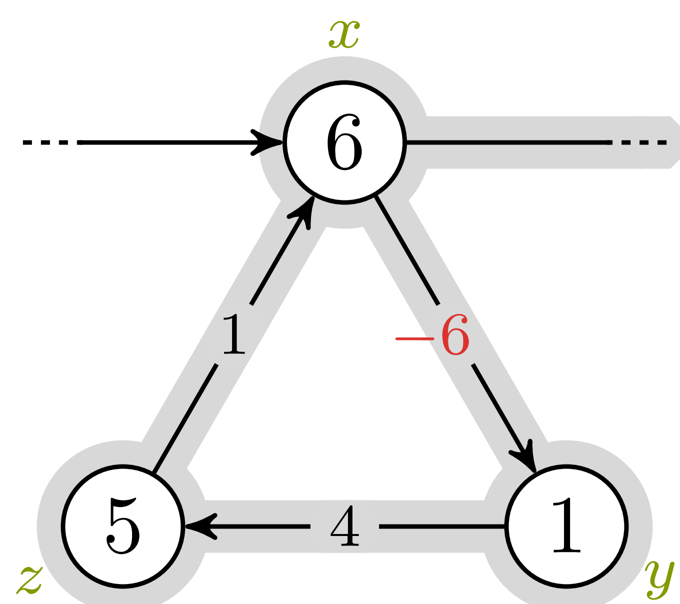
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



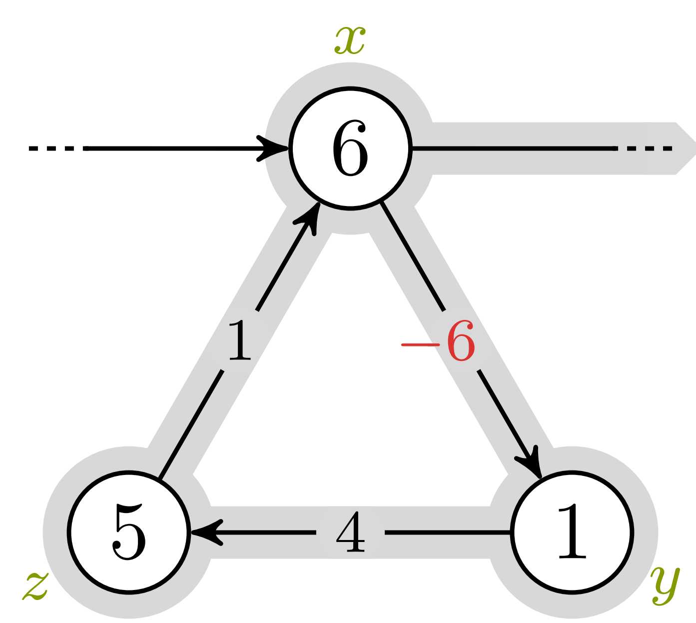
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



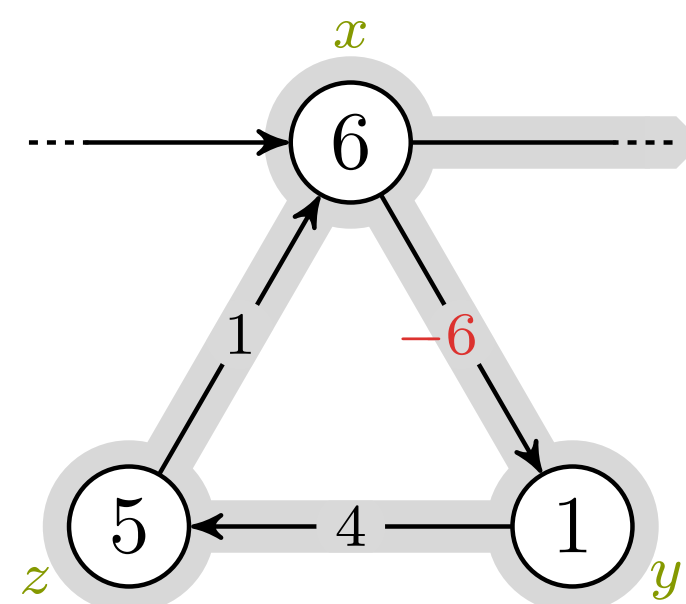
```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```



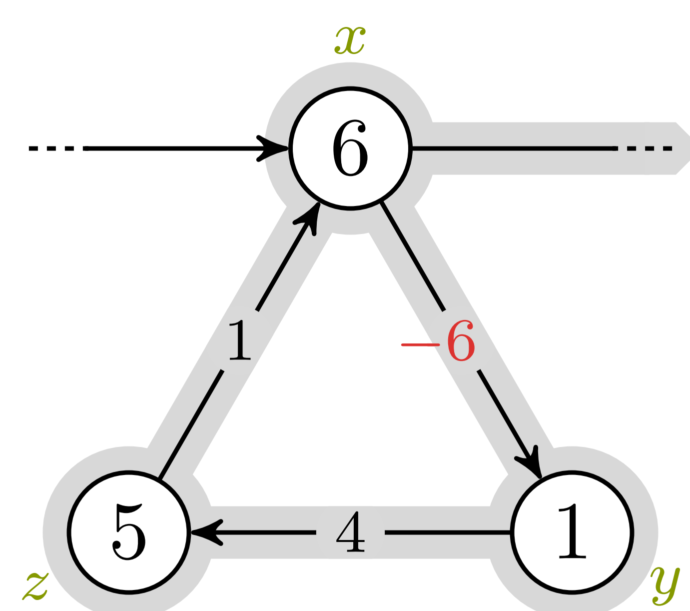
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)



Etc.

```
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
Relax(x, y)
Relax(y, z)
Relax(z, x)
```

Etc.



Ingen sti er kortest!

- **En enkel sti er en sti uten sykler**
- **En kortest sti er alltid enkel**
- **Negativ sykel? Ingen sti er kortest!**
- **Det finnes fortsatt en kortest enkel sti**
- **Å finne den effektivt: Uløst (NP-hardt)**

La $\langle v_1, v_2 \ldots, v_k \rangle$ være korteste vei til z.

La $\langle v_1, v_2 \ldots, v_k \rangle$ være korteste vei til z.

Vi vil slakke kantene langs stien, men kjenner ikke rekkefølgen.

La $\langle v_1, v_2 \ldots, v_k \rangle$ være korteste vei til z.

Vi vil slakke kantene langs stien, men kjenner ikke rekkefølgen.

Løsning:

Slakk absolutt alle kanter $k - 1$ ganger!

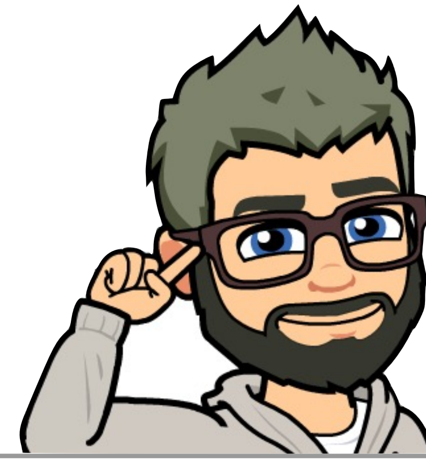
La $\langle v_1, v_2 \ldots, v_k \rangle$ være korteste vei til z.

Vi vil slakke kantene langs stien, men kjenner ikke rekkefølgen.

Løsning:

Slakk absolutt alle kanter $k - 1$ ganger!

Iterasjon i slakker alle, og dermed også (v_i, v_{i+1})



La $\langle v_1, v_2 \ldots, v_k \rangle$ være korteste vei til z.

Vi vil slakke kantene langs stien, men kjenner ikke rekkefølgen.

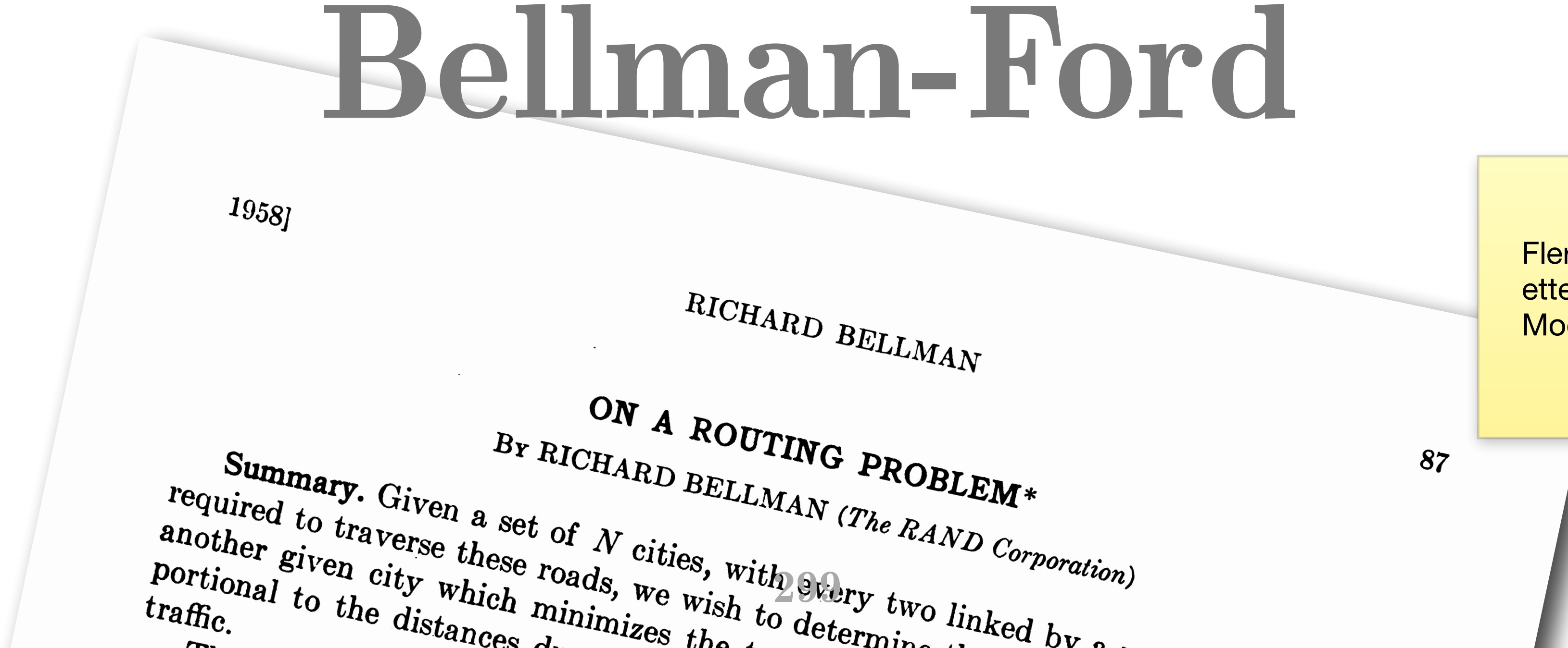
Løsning:

Slakk absolutt alle kanter $k - 1$ ganger!

Hvis vi lar $k = |V|$ så får alle noder rett estimat

4:5

Bellman-Ford


1958]

RICHARD BELLMAN

87

ON A ROUTING PROBLEM*

By RICHARD BELLMAN (*The RAND Corporation*)

Summary. Given a set of N cities, with every two linked by a
required to traverse these roads, we wish to determine the
another given city which minimizes the
portional to the distances
traffic.

Flere har publisert metoden før/ etter, inkl. Shimbel, Ford og Moore.

$\textsc{Bellman-Ford}(\mathrm{G}, w, s)$

G	graf
w	vekting
s	startnode

Slakk alle kantene til det bare <u>må</u> bli rett!

Bellman-Ford(G, w, s)
1 Initialize-Single-Source(G, s)

G graf
w vekting
s startnode

BELLMAN-FORD(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 **for** $i = 1$ **to** $|\mathrm{G.V}| - 1$

G	graf
w	vekting
s	startnode
i	teller

Ingen stier har flere enn $|V| - 1$ kanter

Bellman-Ford(G, w, s)
1 Initialize-Single-Source(G, s)
2 **for** $i = 1$ **to** $|G.V| - 1$
3 **for** each edge $(u, v) \in G.E$

G	graf
w	vekting
s	startnode
i	teller
u	fra-node
v	til-node

Gå gjennom alle $|V| - 1$ ganger

Bellman-Ford(G, w, s)

```
1  Initialize-Single-Source(G, s)
2  for i = 1 to |G.V| - 1
3      for each edge (u, v) ∈ G.E
4          Relax(u, v, w)
```

G graf
w vekting
s startnode
i teller
u fra-node
v til-node

Alle stier må nå ha fått kantene slakket i rekkefølge!

Bellman-Ford(G, w, s)

Initialize-Single-Source(G, s)
for $i = 1$ **to** $|G.V| - 1$
    for each edge $(u, v) \in G.E$
        Relax(u, v, w)
for each edge $(u, v) \in G.E$

G	graf
w	vekting
s	startnode
i	teller
u	fra-node
v	til-node

Vi kjører én iterasjon til…

Bellman-Ford(G, w, s)

Initialize-Single-Source(G, s)
for $i = 1$ **to** $|G.V| - 1$
    for each edge $(u, v) \in G.E$
        Relax(u, v, w)
for each edge $(u, v) \in G.E$
    if $v.d > u.d + w(u, v)$

G	graf
w	vekting
s	startnode
i	teller
u	fra-node
v	til-node

Men bare sjekker om det fortsatt finnes snarveier

BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for $i = 1$ **to** $|G.V| - 1$
  for each edge $(u, v) \in G.E$
    RELAX(u, v, w)
for each edge $(u, v) \in G.E$
  if $v.d > u.d + w(u, v)$
    return FALSE

G graf
w vekting
s startnode
i teller
u fra-node
v til-node

I så fall: Vi må ha kommet borti en negativ sykel

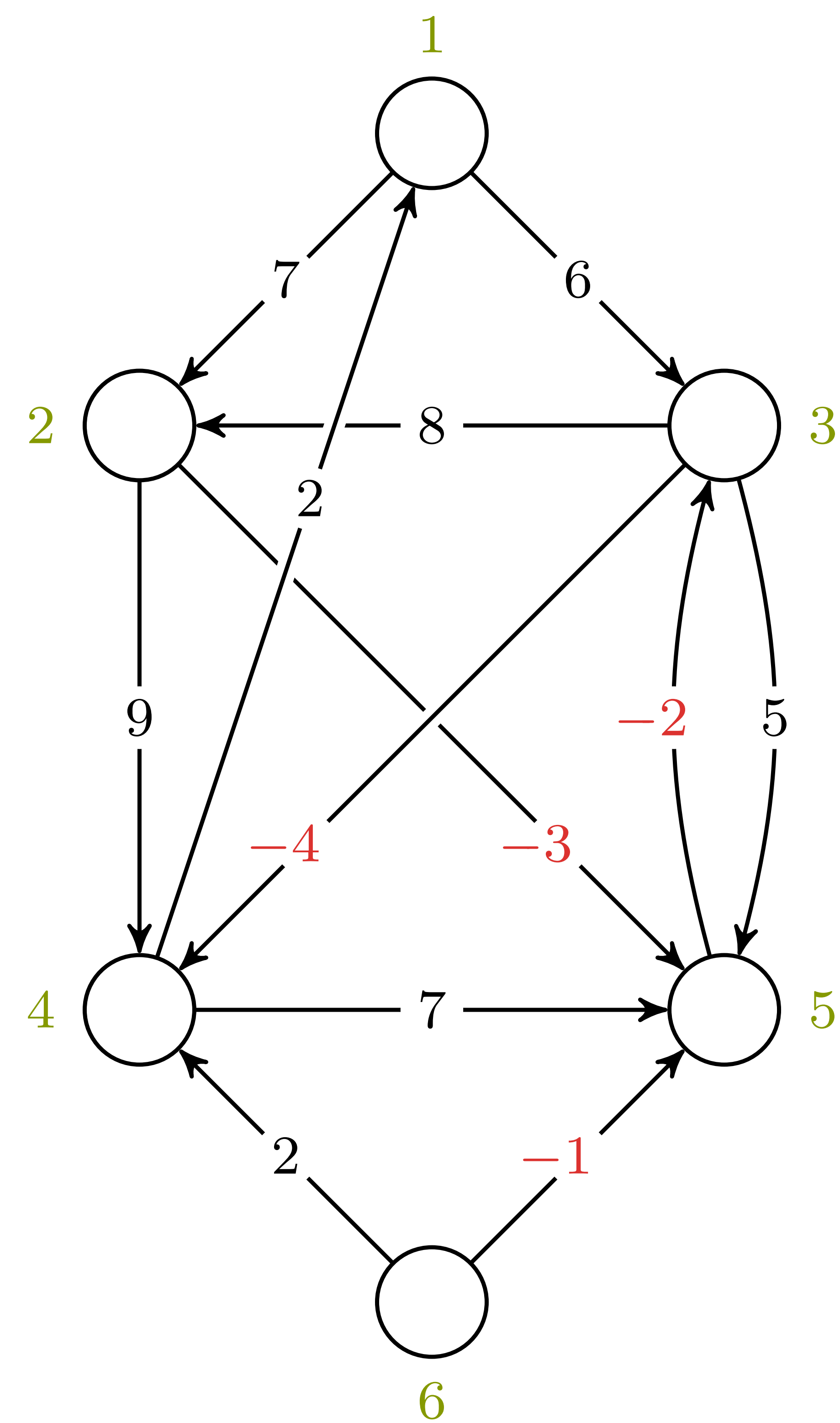
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

G graf
w vekting
s startnode
i teller
u fra-node
v til-node

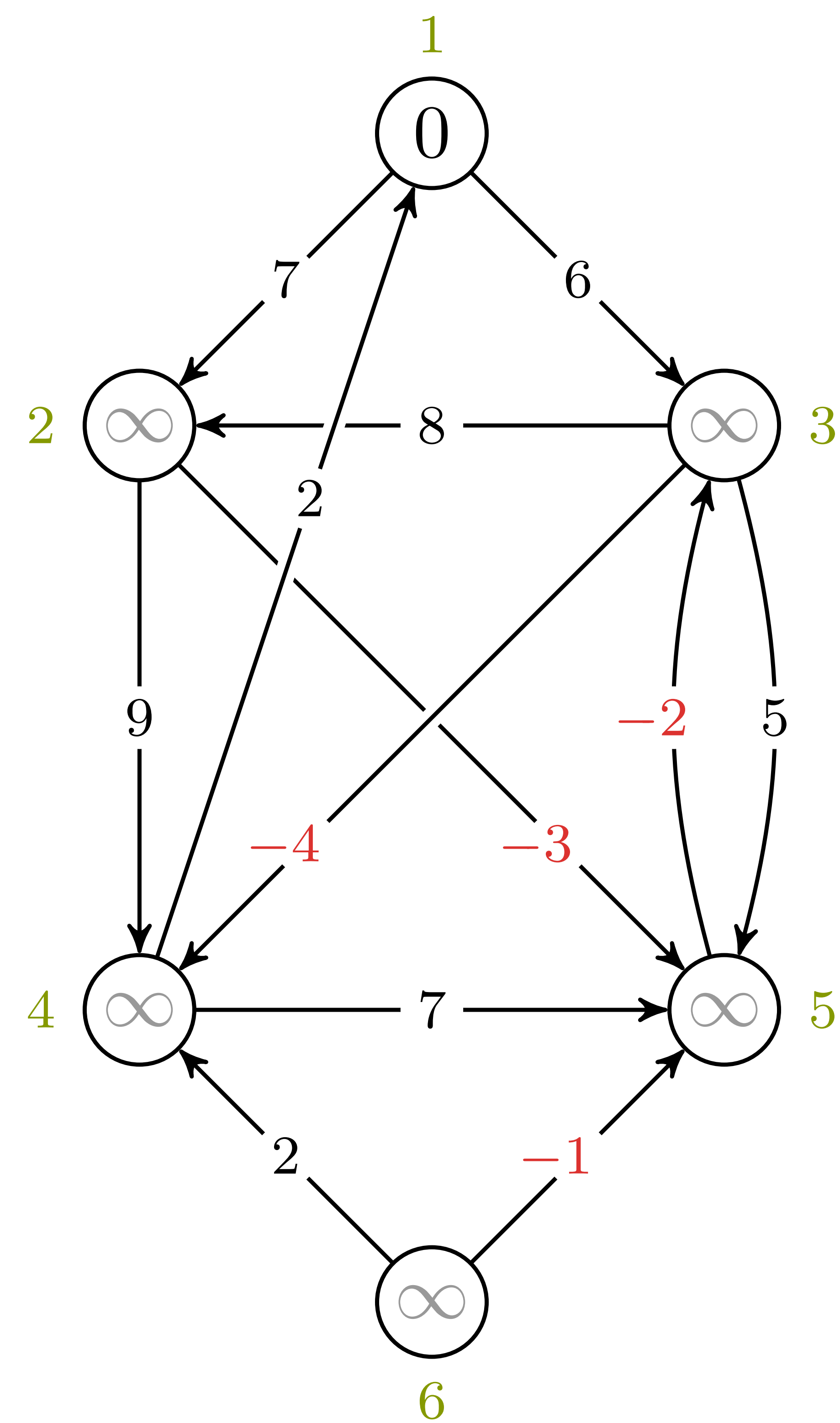
Ellers: Svaret vi fant må være rett!

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

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



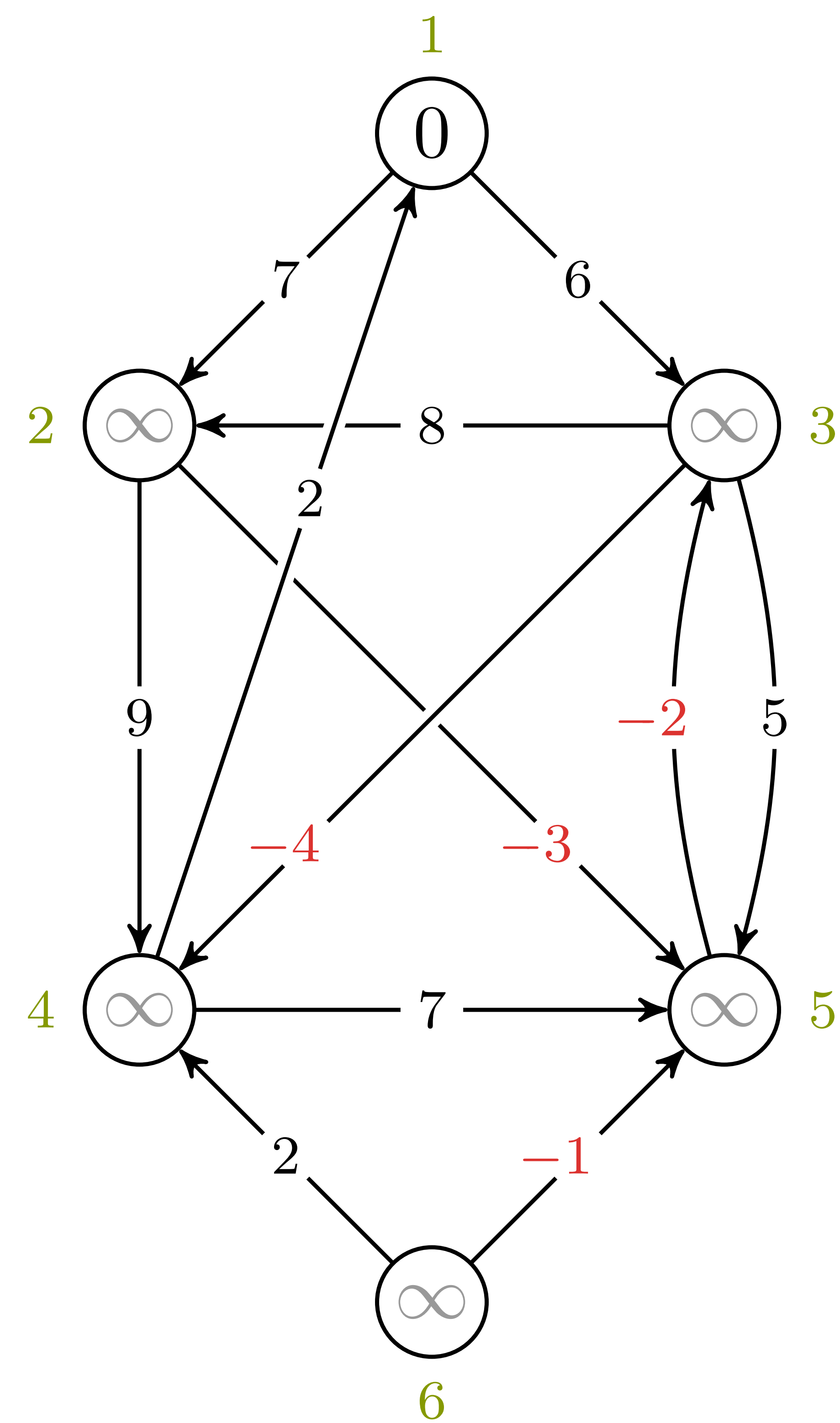
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



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

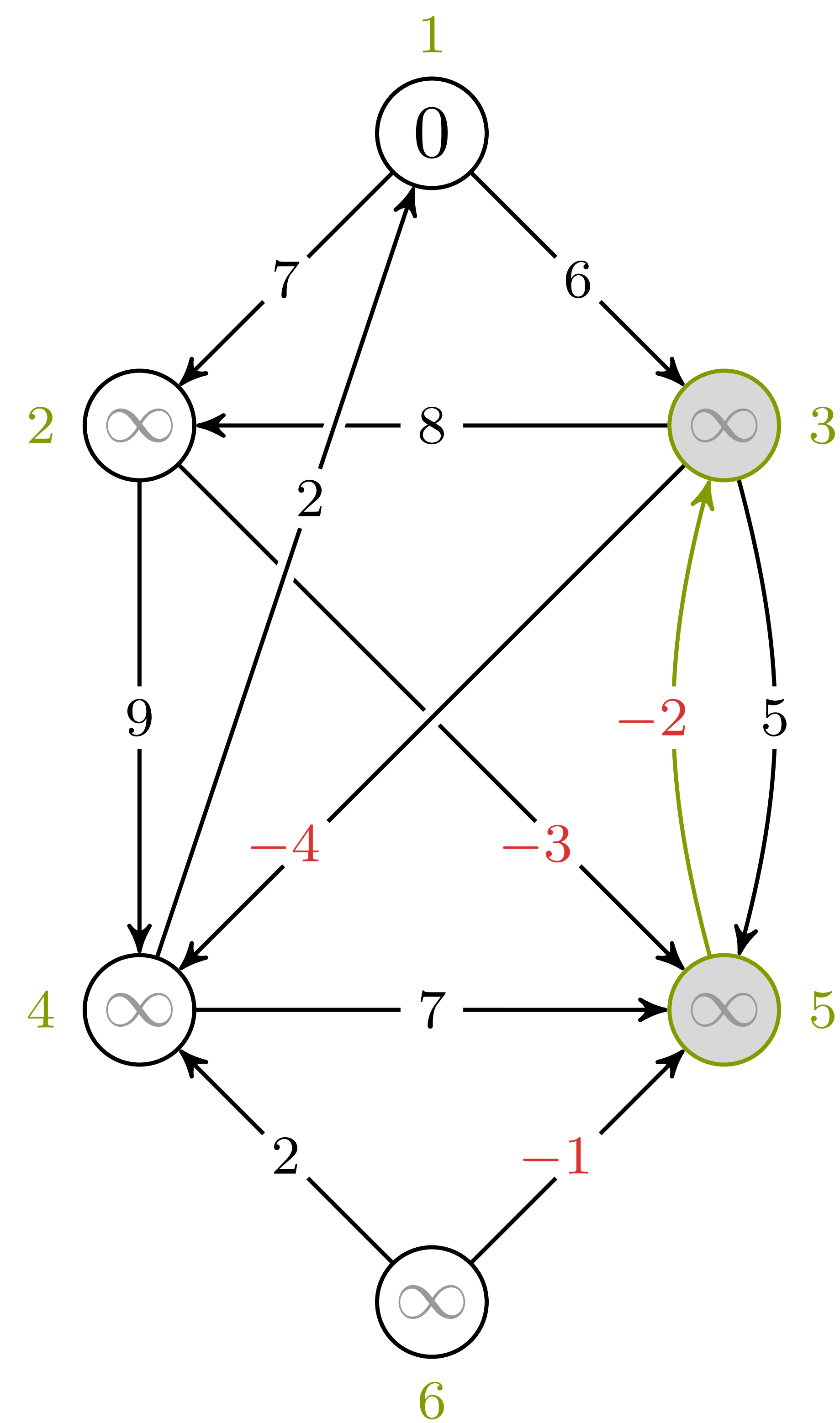
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

$i, u, v \ = \ 1, -, -$

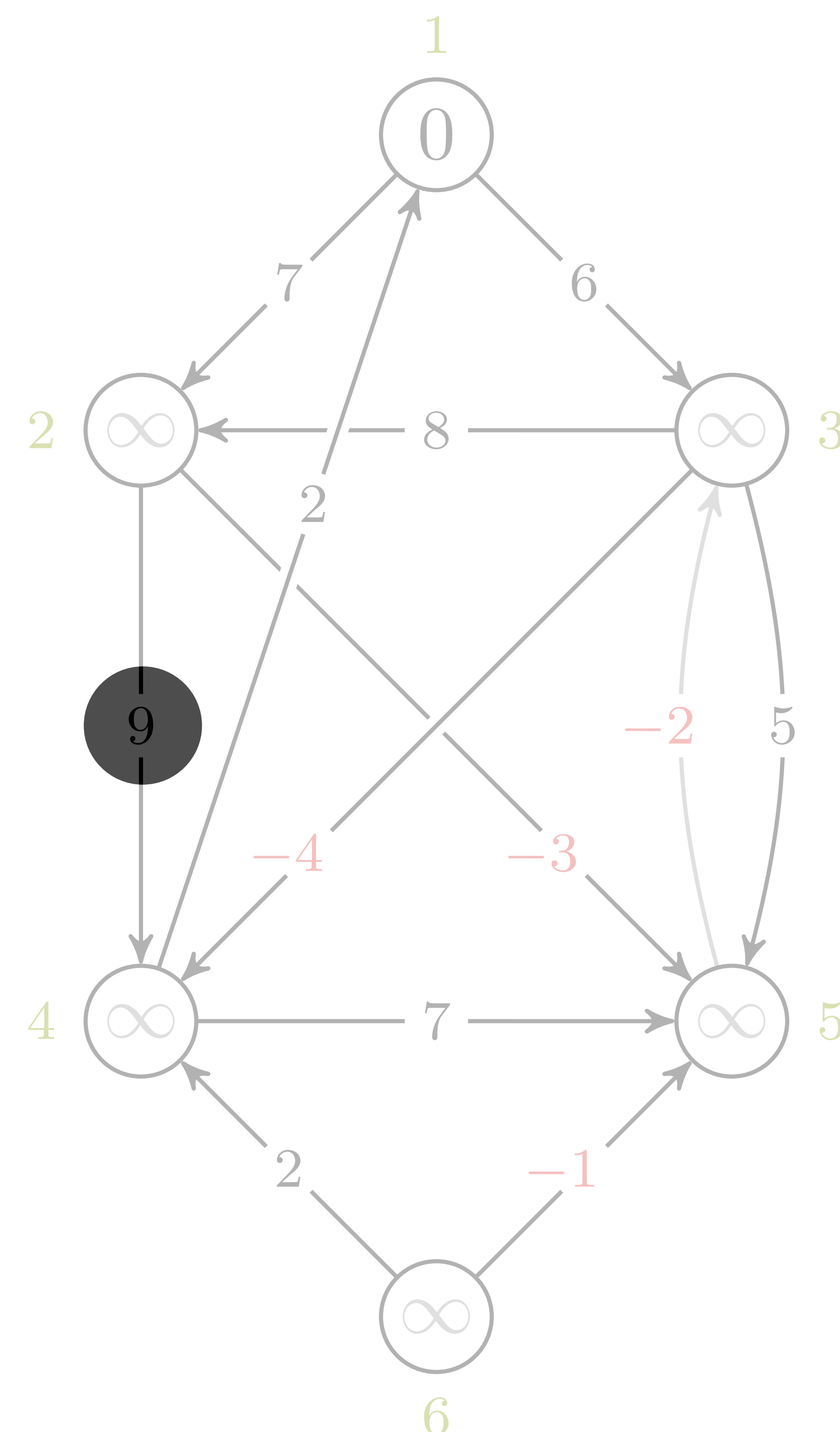


```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

$i, u, v \ = \ 1, 5, 3$

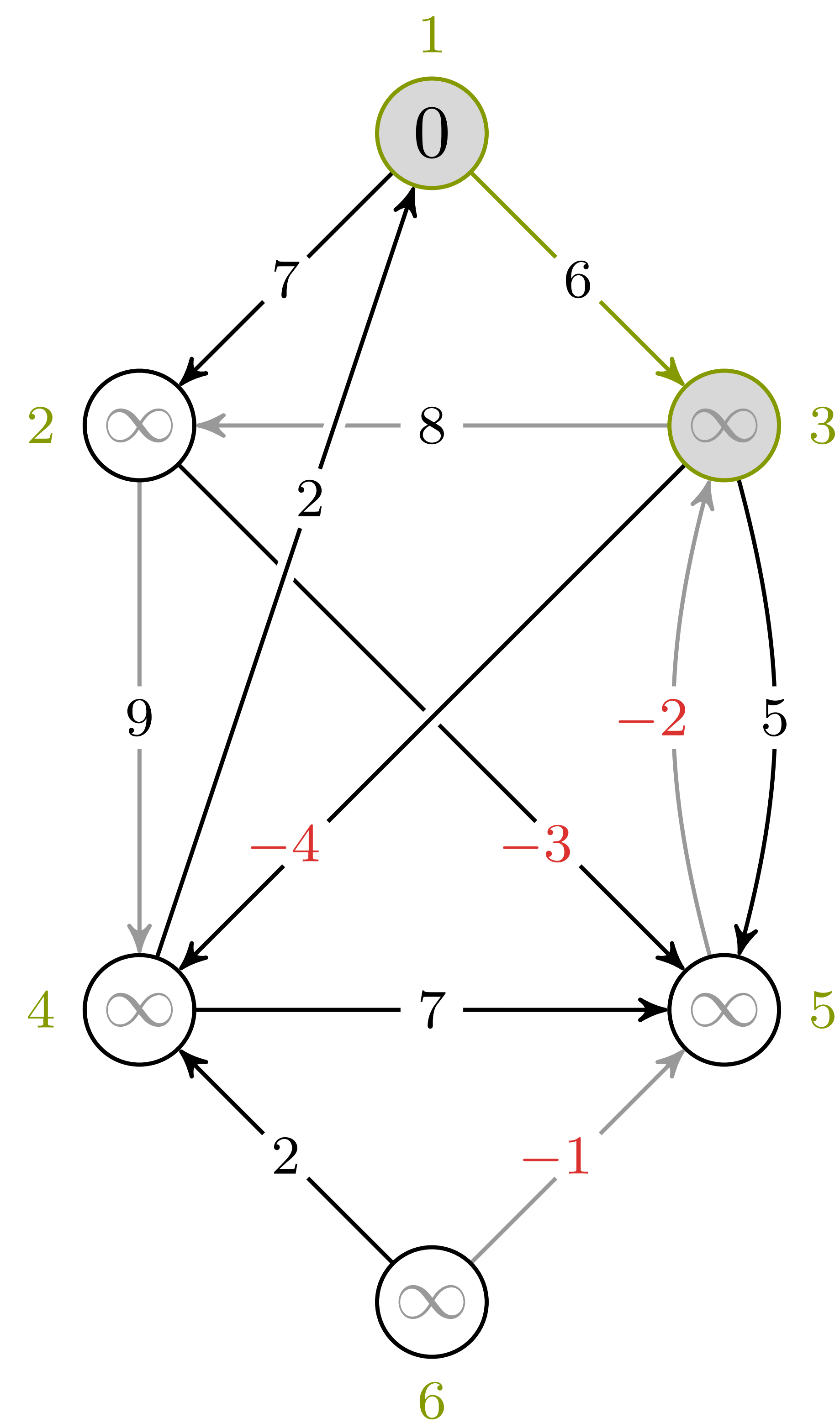


```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          Relax(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return false
  return true
```



$i, u, v \ = \ 1, 5, 3$

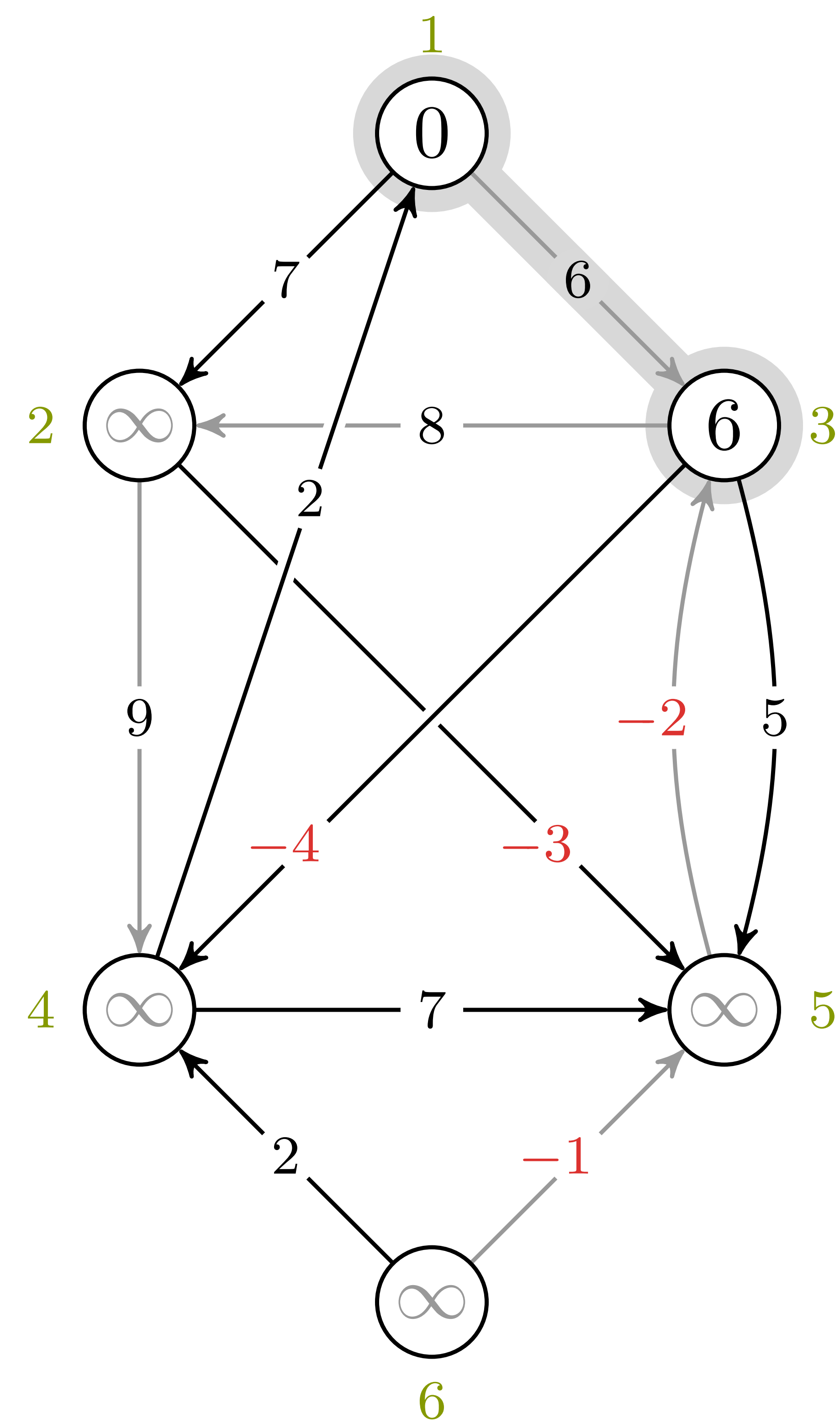
```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          Relax(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return false
  return true
```



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

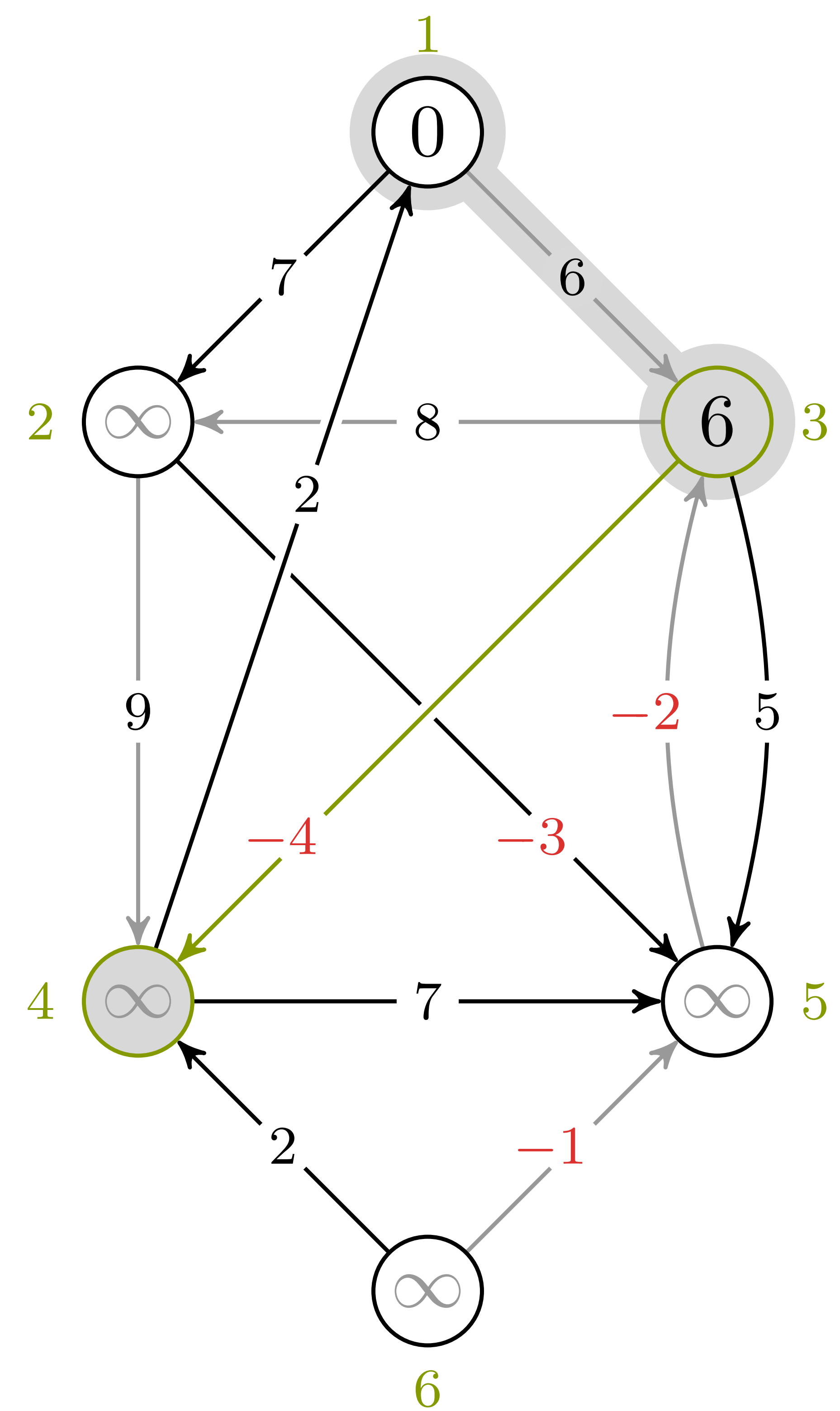
```
Bellman-Ford(G, w, s)
Initialize-Single-Source(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        Relax(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return false
return true
```

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

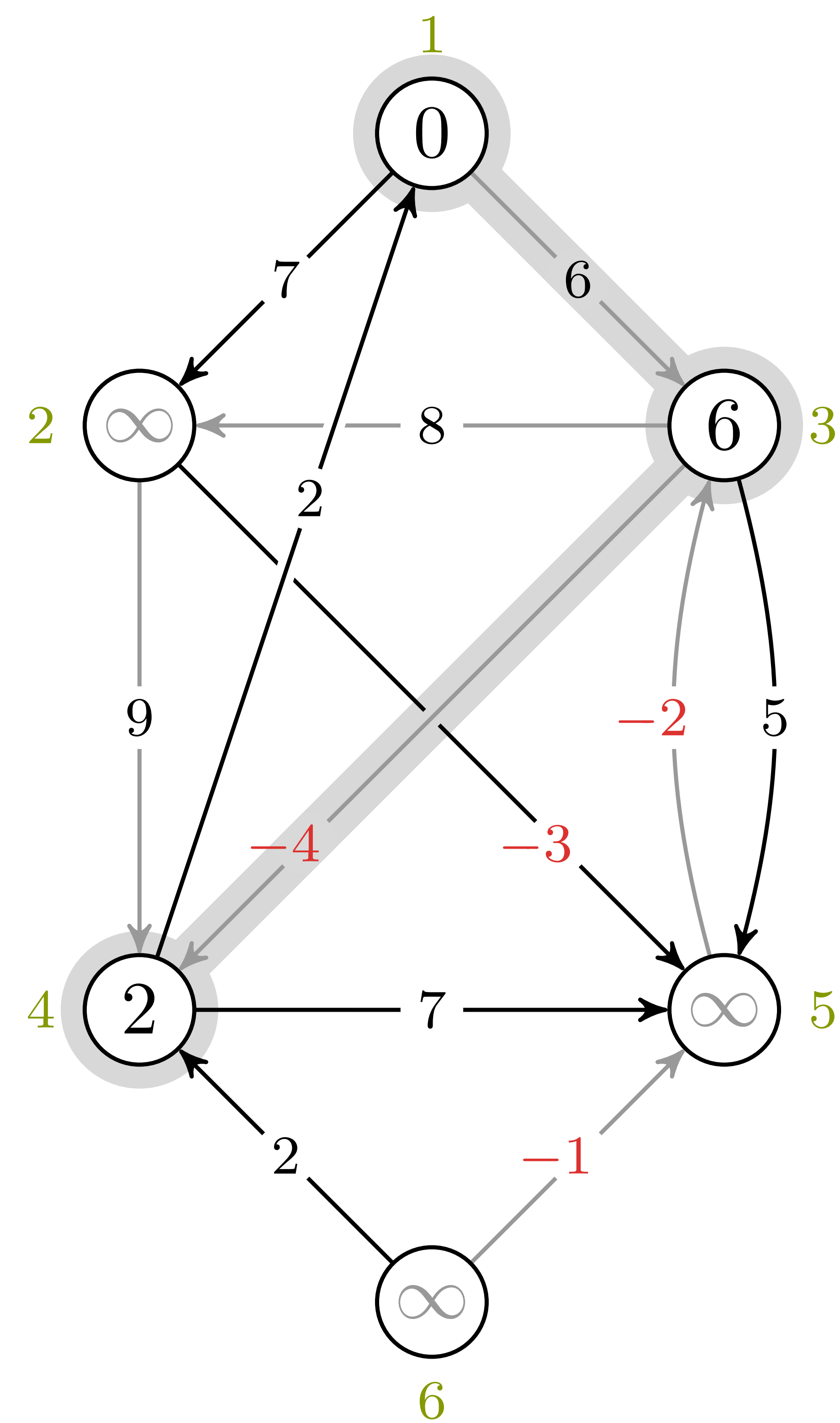


```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

$i, u, v \;=\; 1, 3, 4$



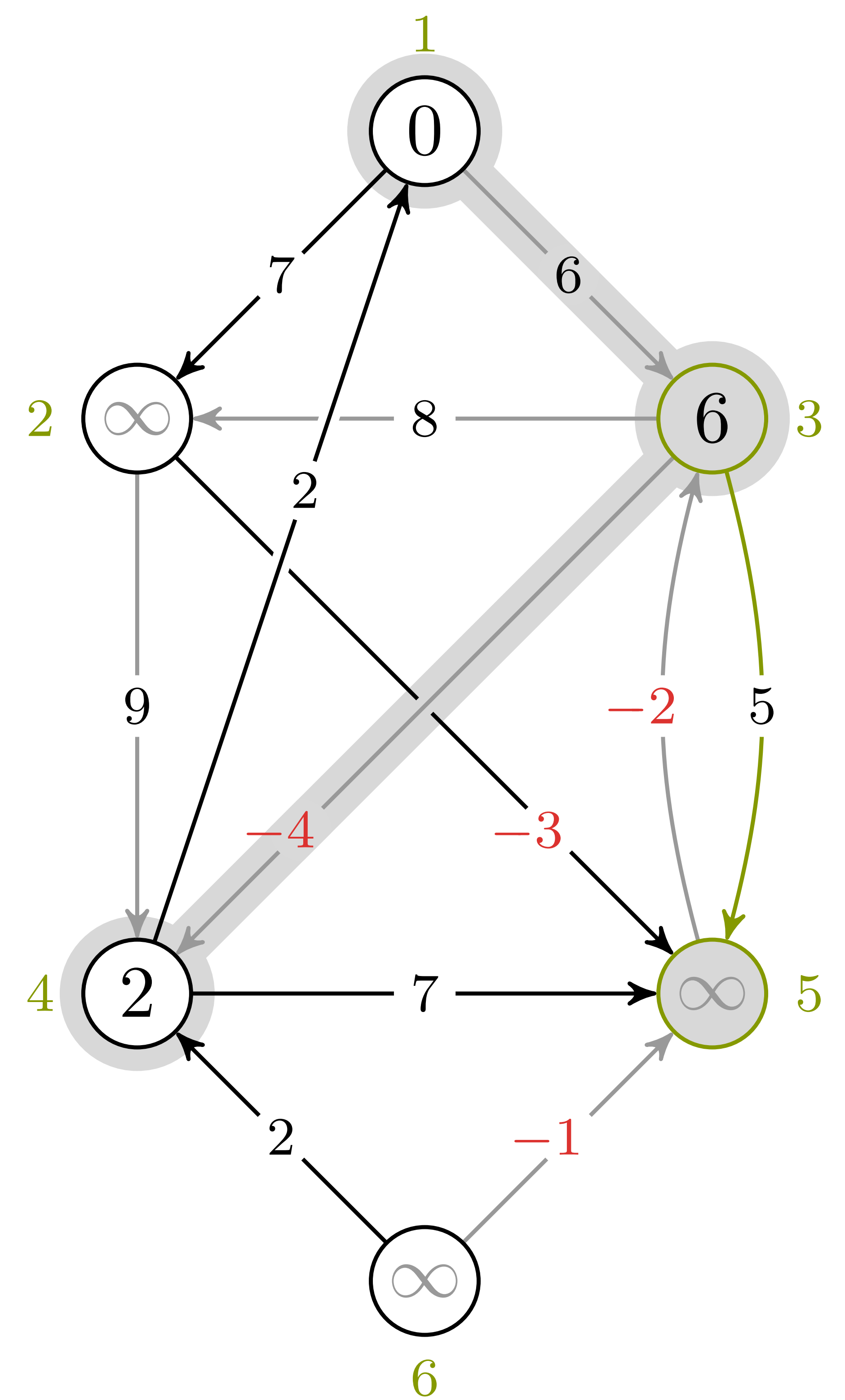
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



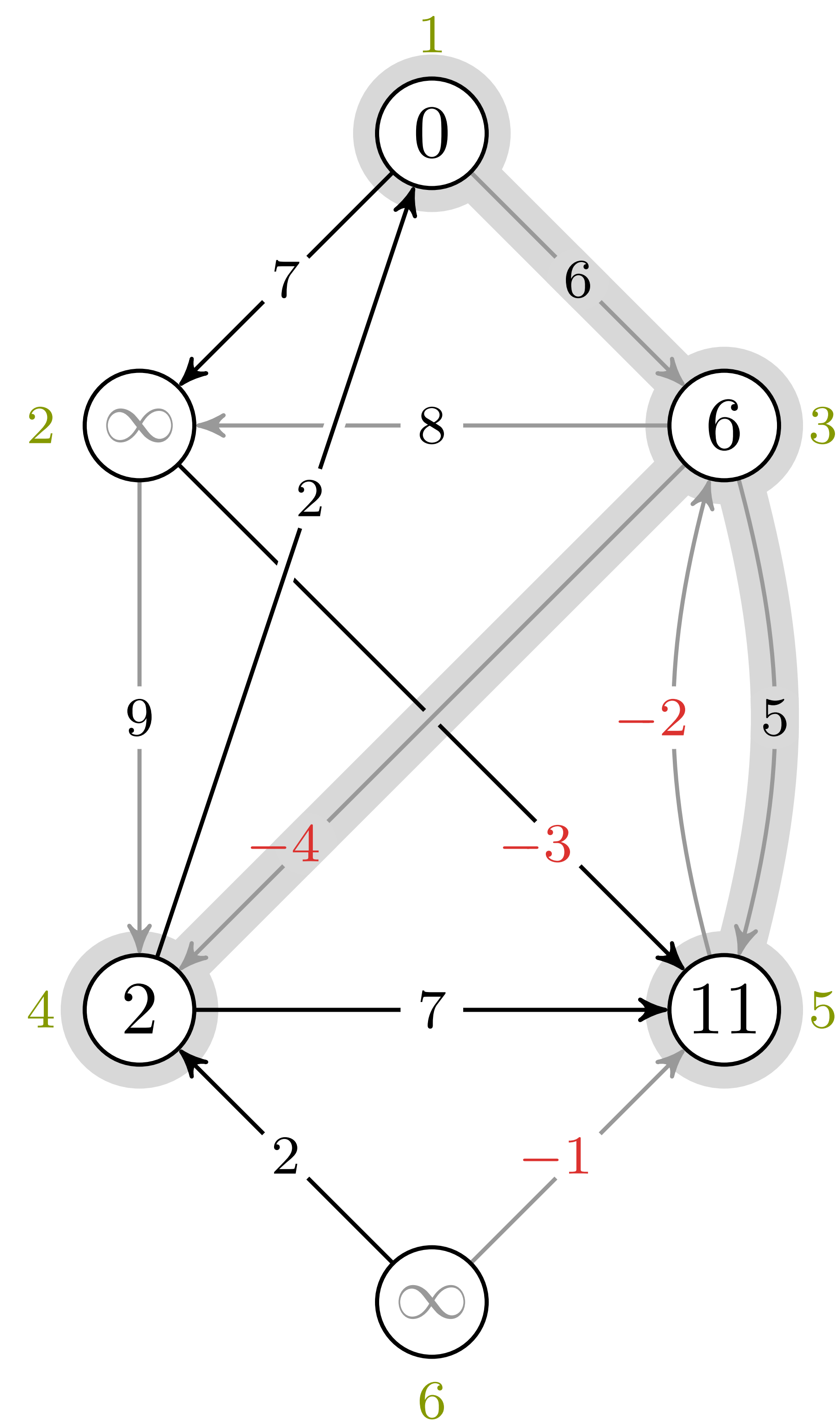
$i, u, v \;=\; 1, 3, 4$

```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

$i, u, v \;=\; 1, 3, 5$

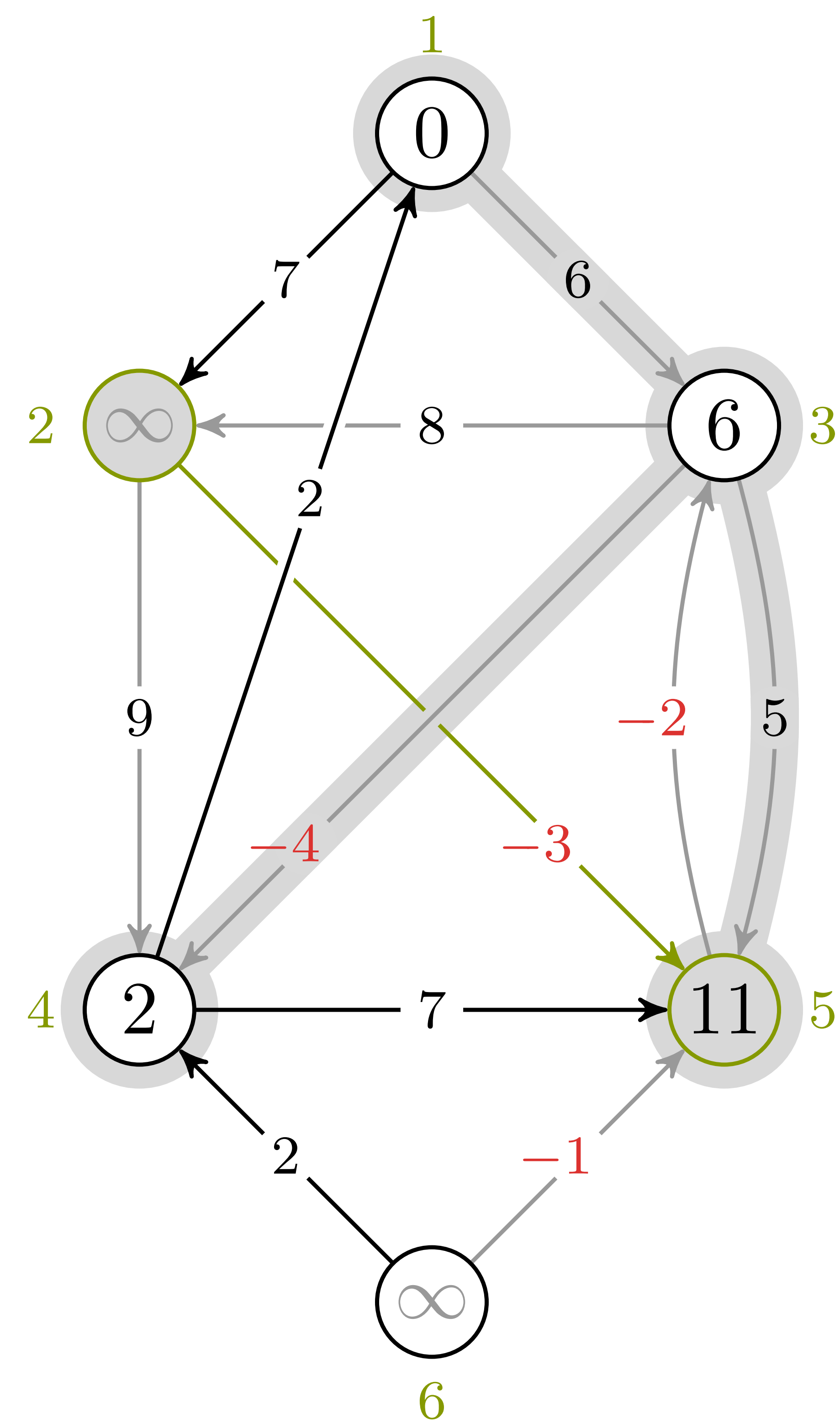


```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



$i, u, v \;=\; 1, 3, 5$

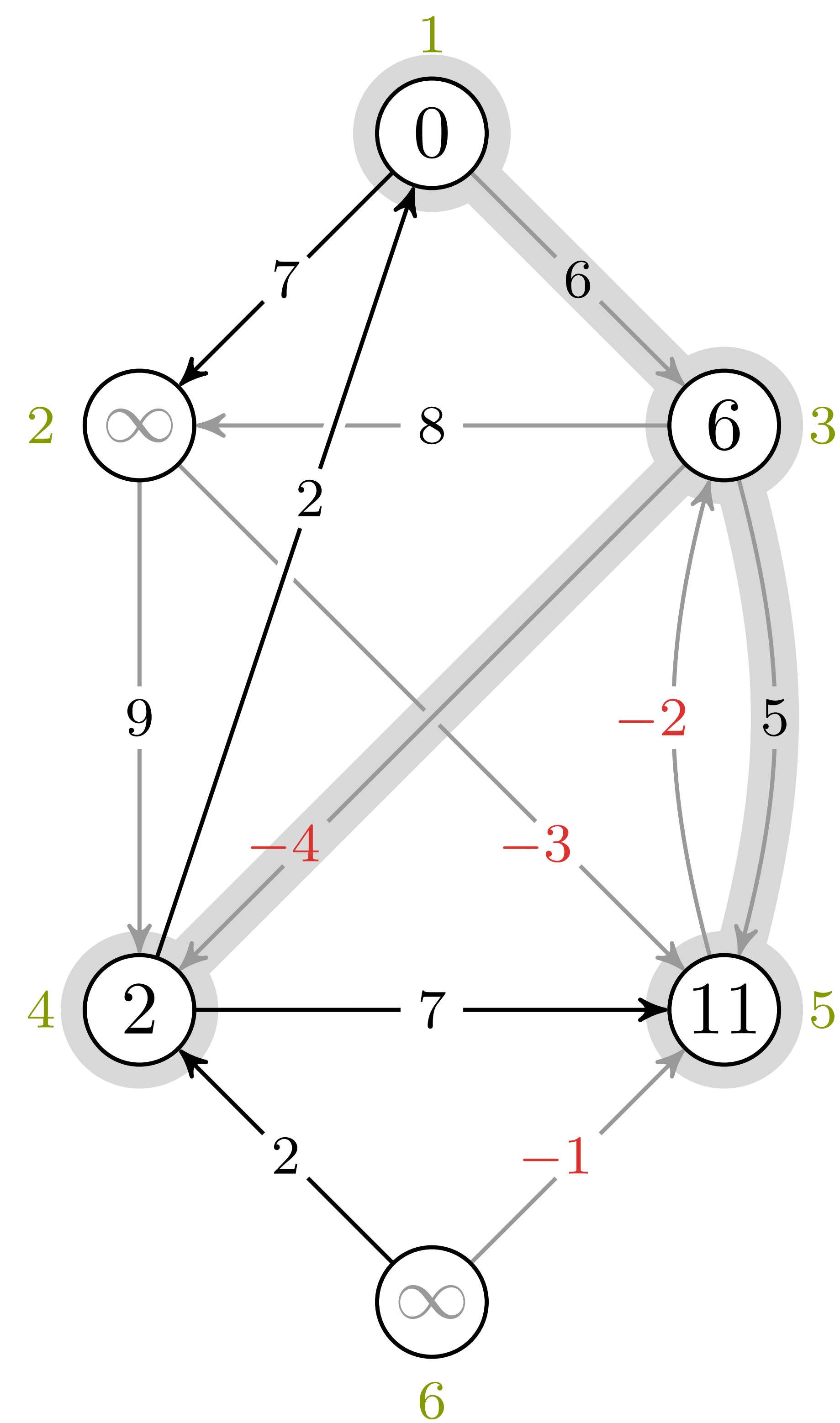
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



$i, u, v \ = \ 1, 2, 5$

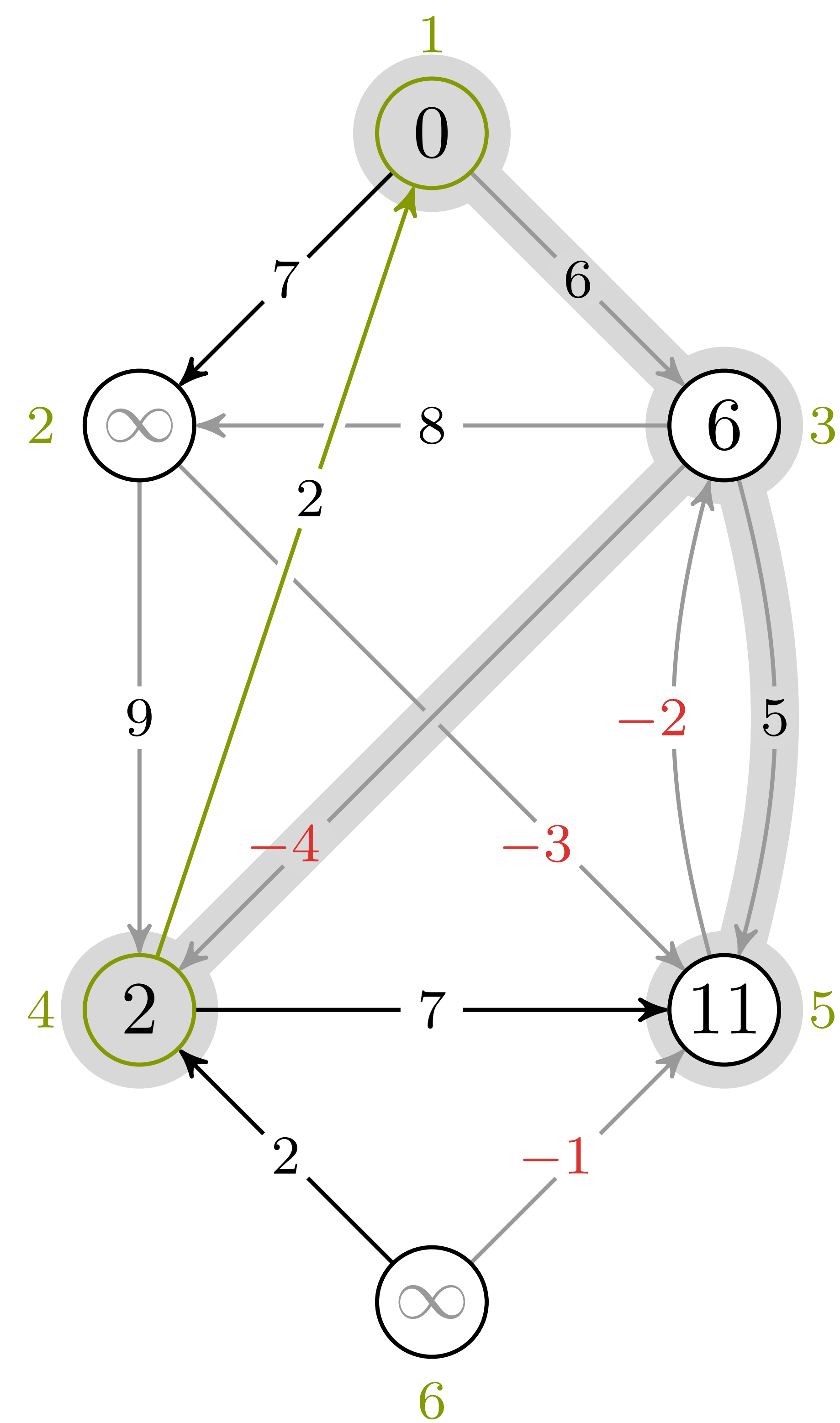
```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          Relax(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

$i, u, v \;=\; 1, 2, 5$



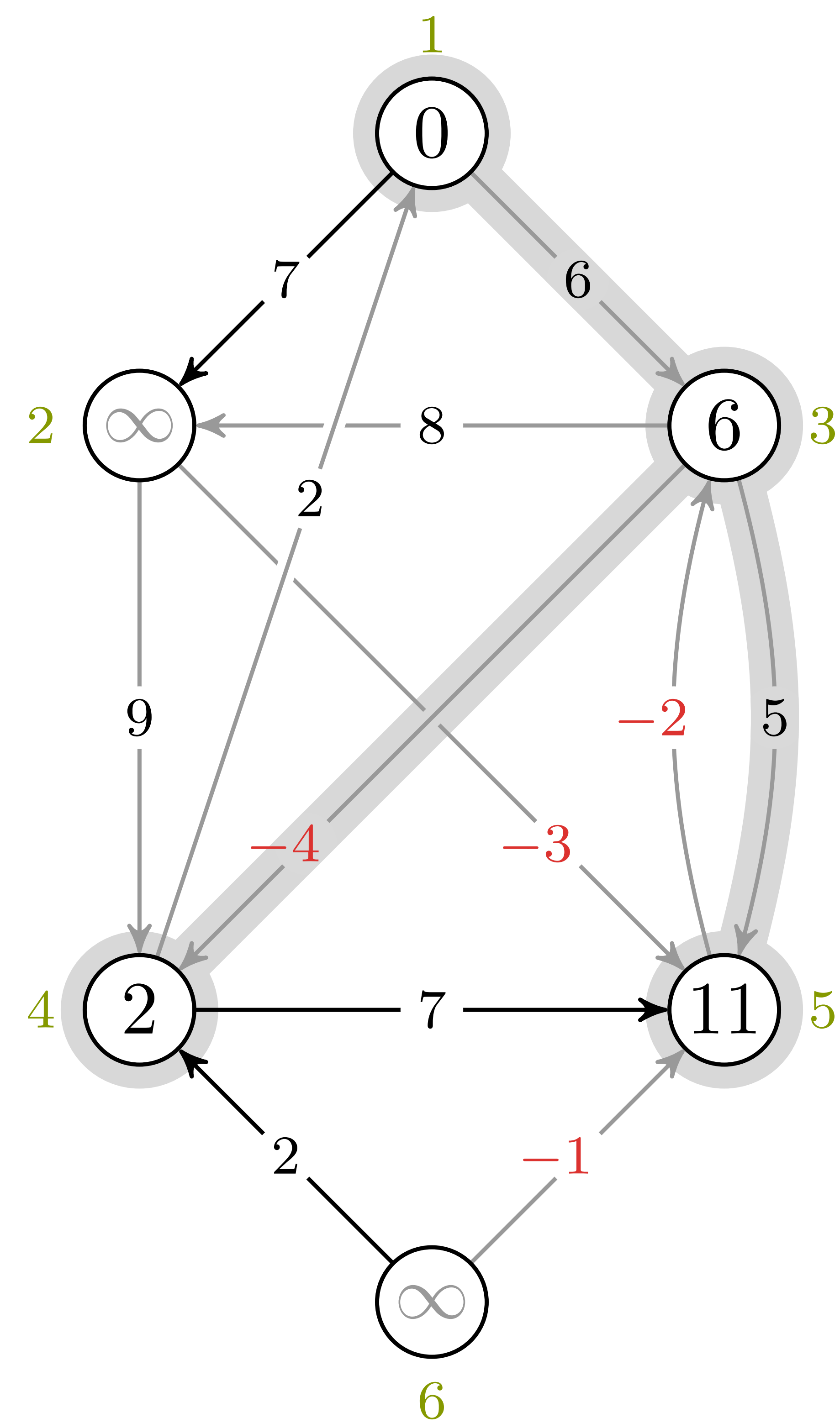
```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i = 1 to |G.V| − 1
        for each edge (u, v) ∈ G.E
            RELAX(u, v, w)
    for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
            return FALSE
    return TRUE
```

$i, u, v \;=\; 1, 4, 1$

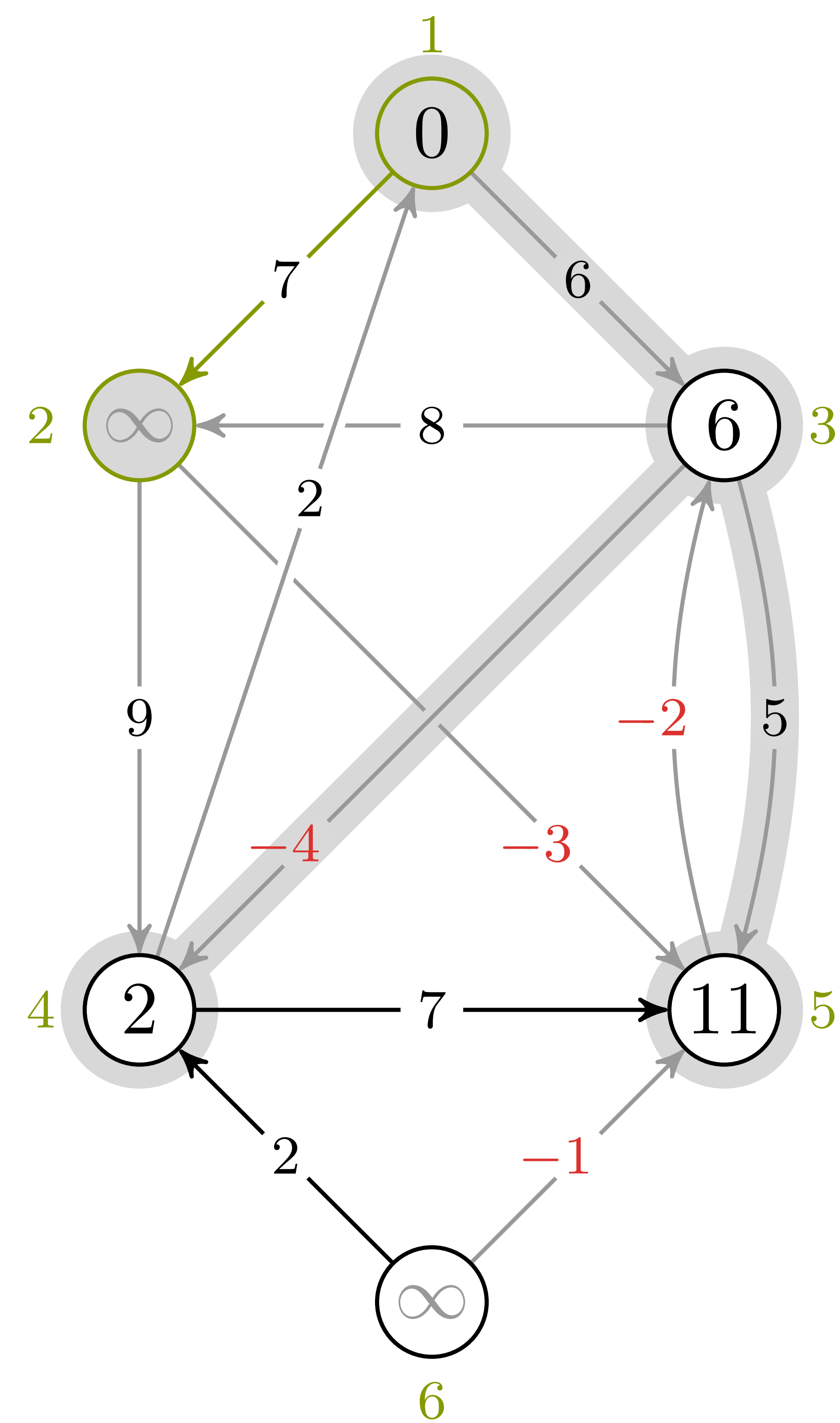


```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          Relax(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return false
  return true
```

$i, u, v \;=\; 1, 4, 1$



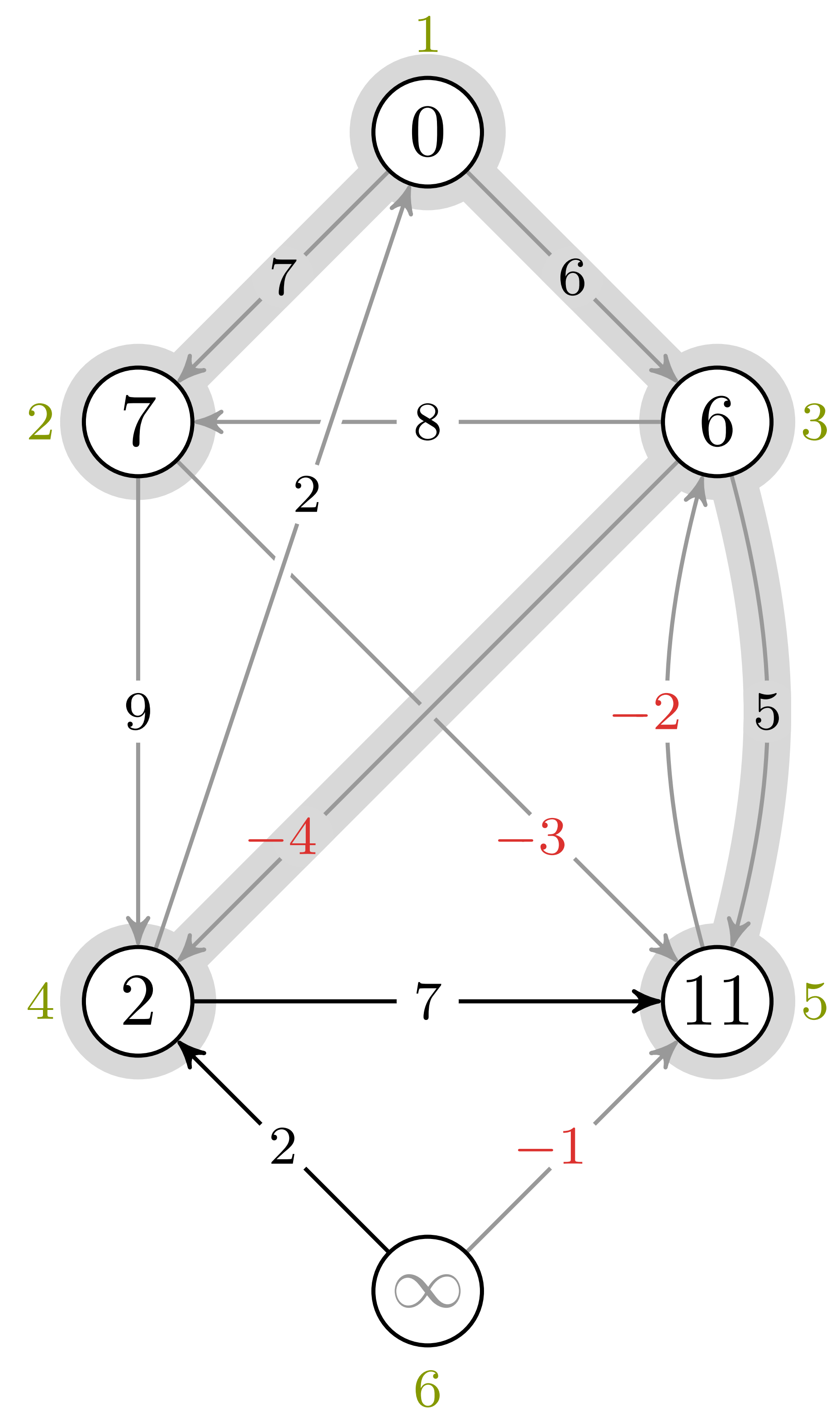
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



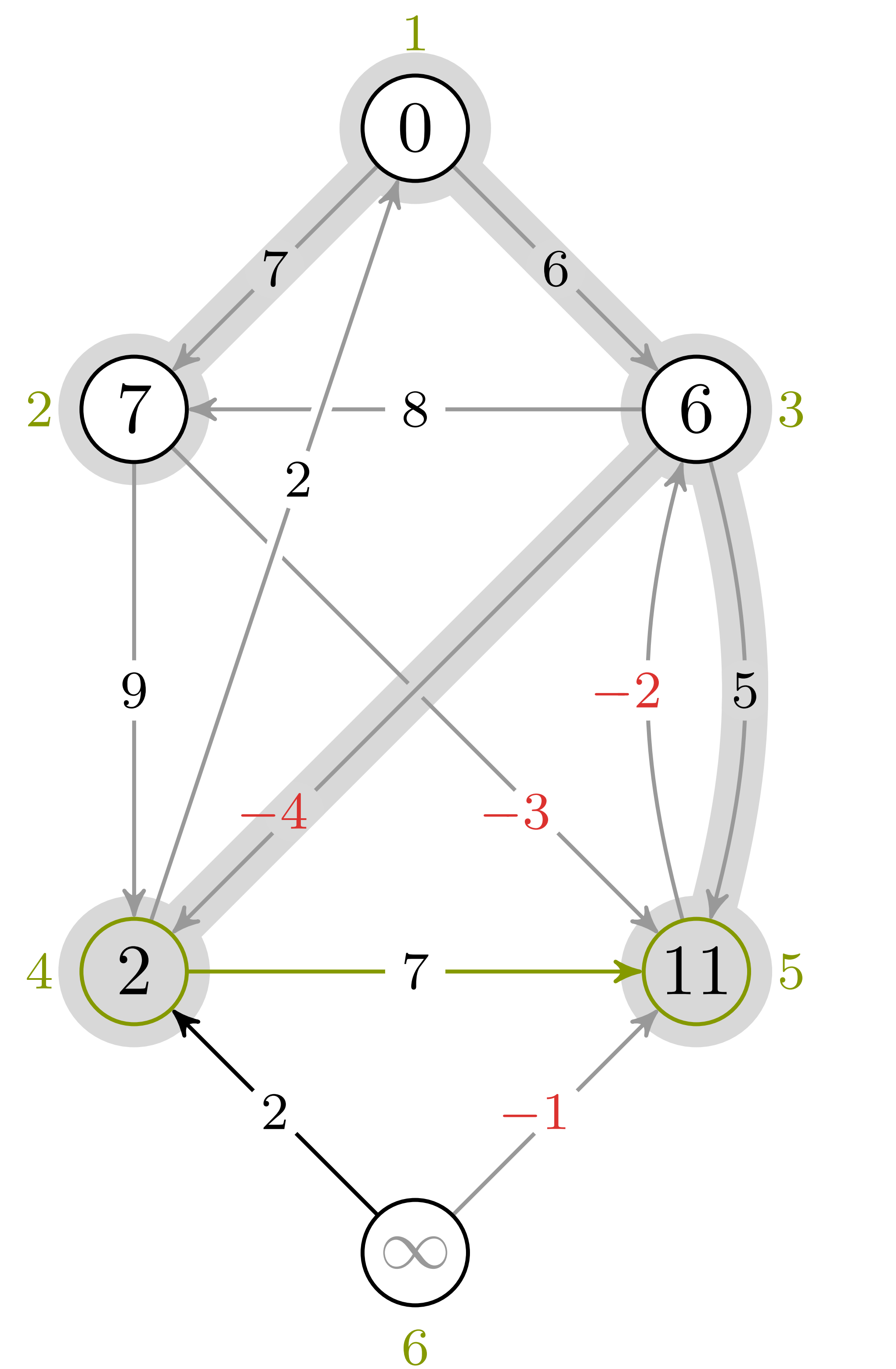
$i, u, v \;=\; 1, 1, 2$

```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i = 1 to |G.V| − 1
        for each edge (u, v) ∈ G.E
            RELAX(u, v, w)
    for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
            return FALSE
    return TRUE
```

$i, u, v \;=\; 1, 1, 2$



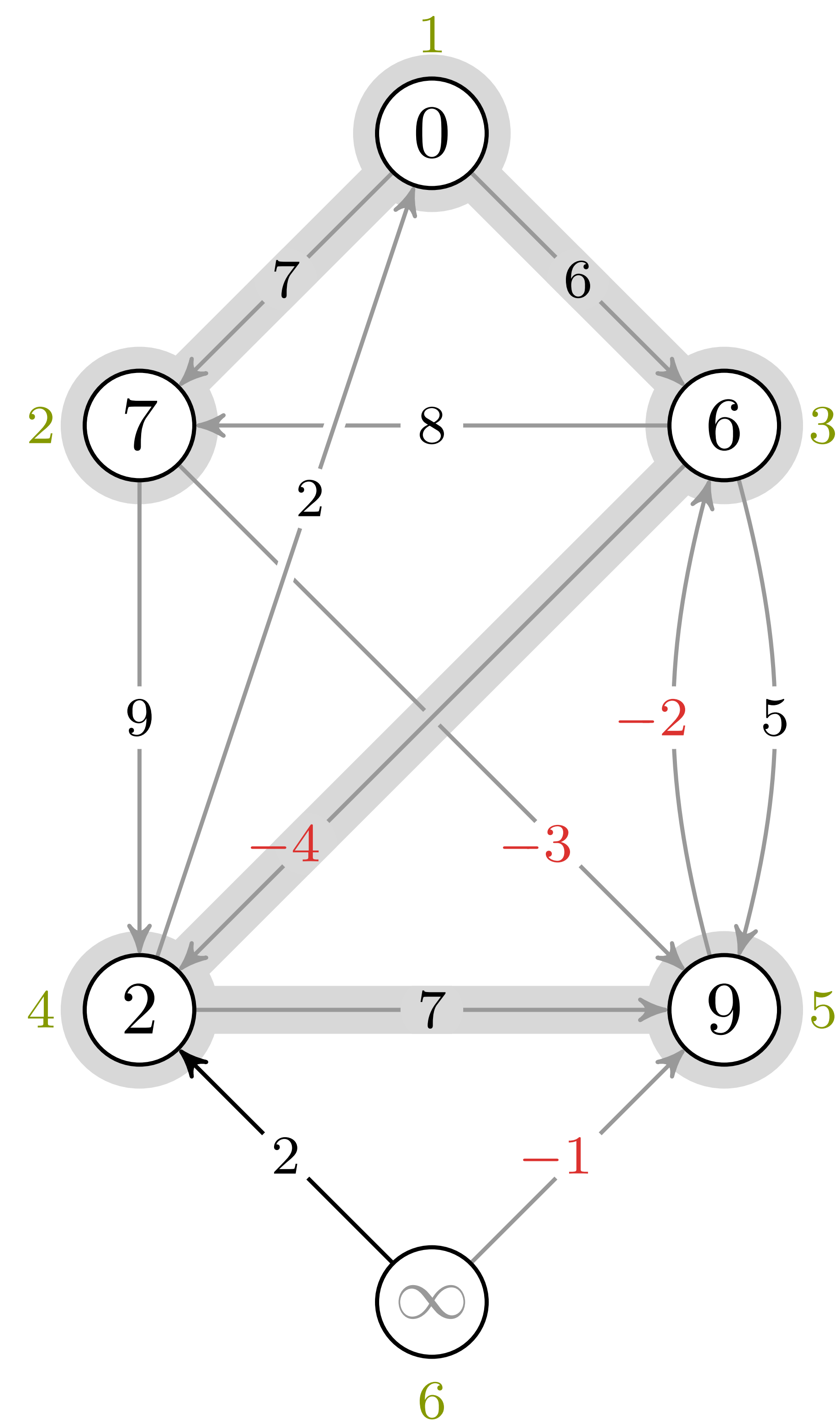
```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i = 1 to |G.V| - 1
        for each edge (u, v) ∈ G.E
            RELAX(u, v, w)
    for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
            return FALSE
    return TRUE
```



$i, u, v \; = \; 1, 4, 5$

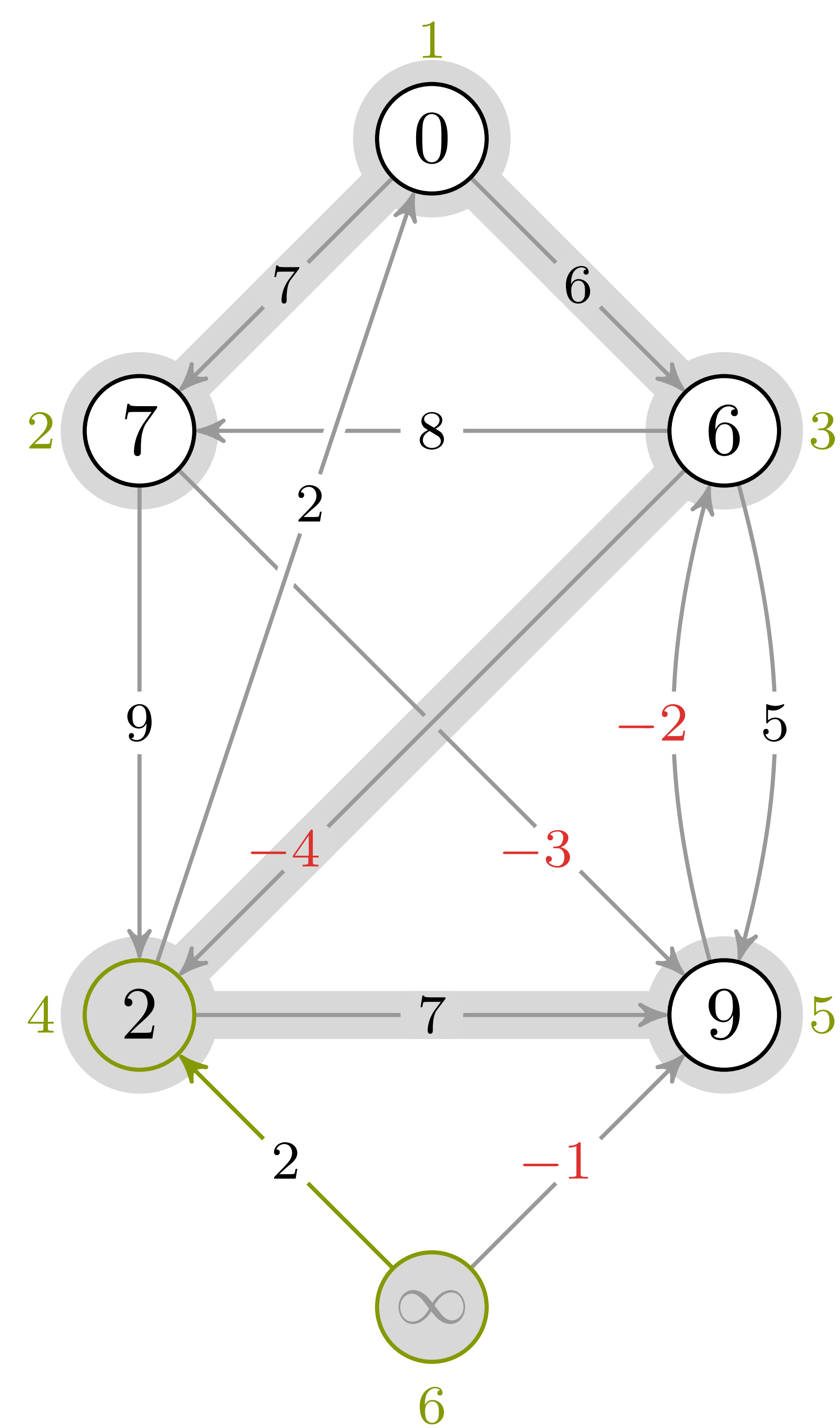
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

$i, u, v \;=\; 1, 4, 5$

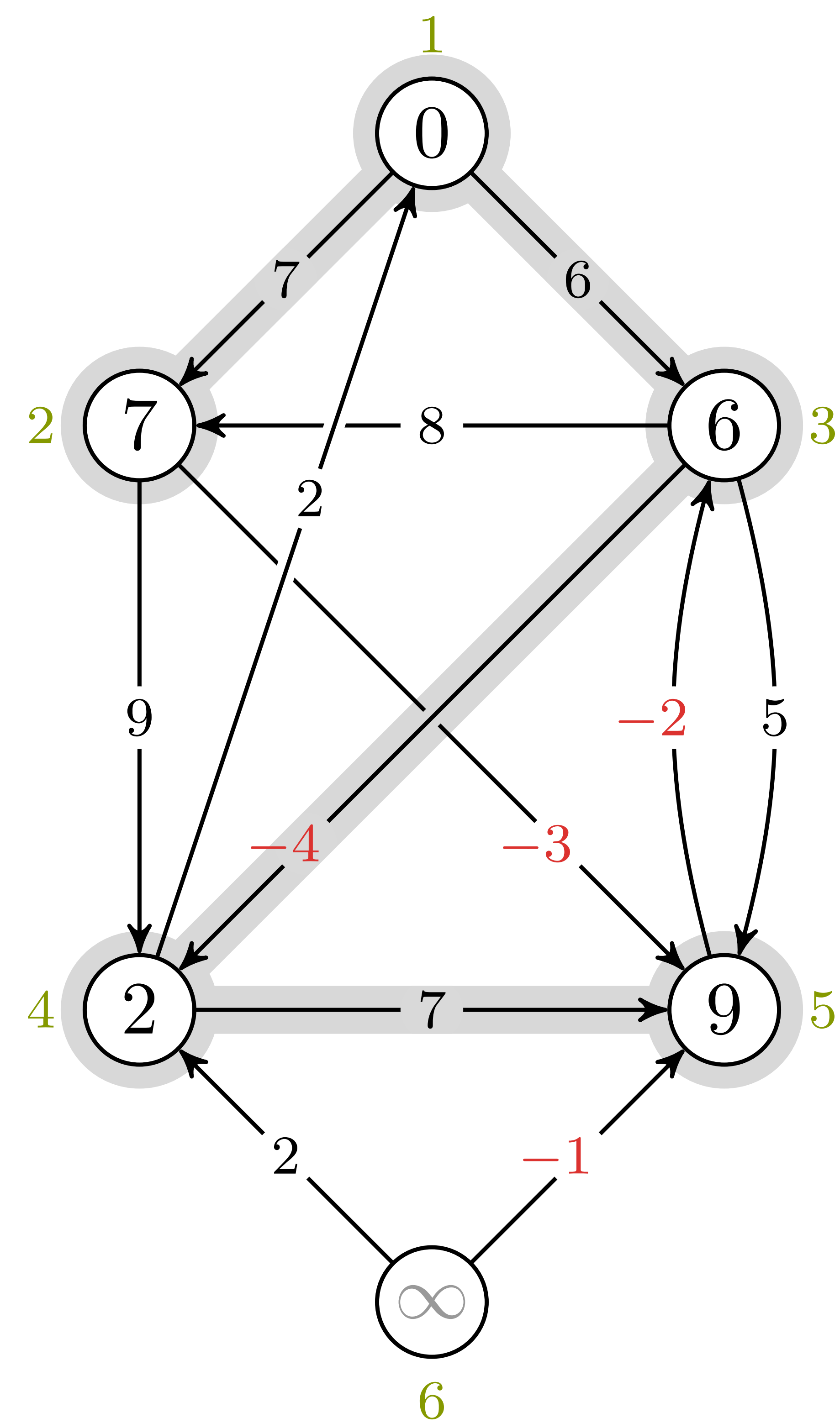


```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
        for each edge (u, v) ∈ G.E
              RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
              return FALSE
  return TRUE
```

$i, u, v \ = \ 1, 6, 4$

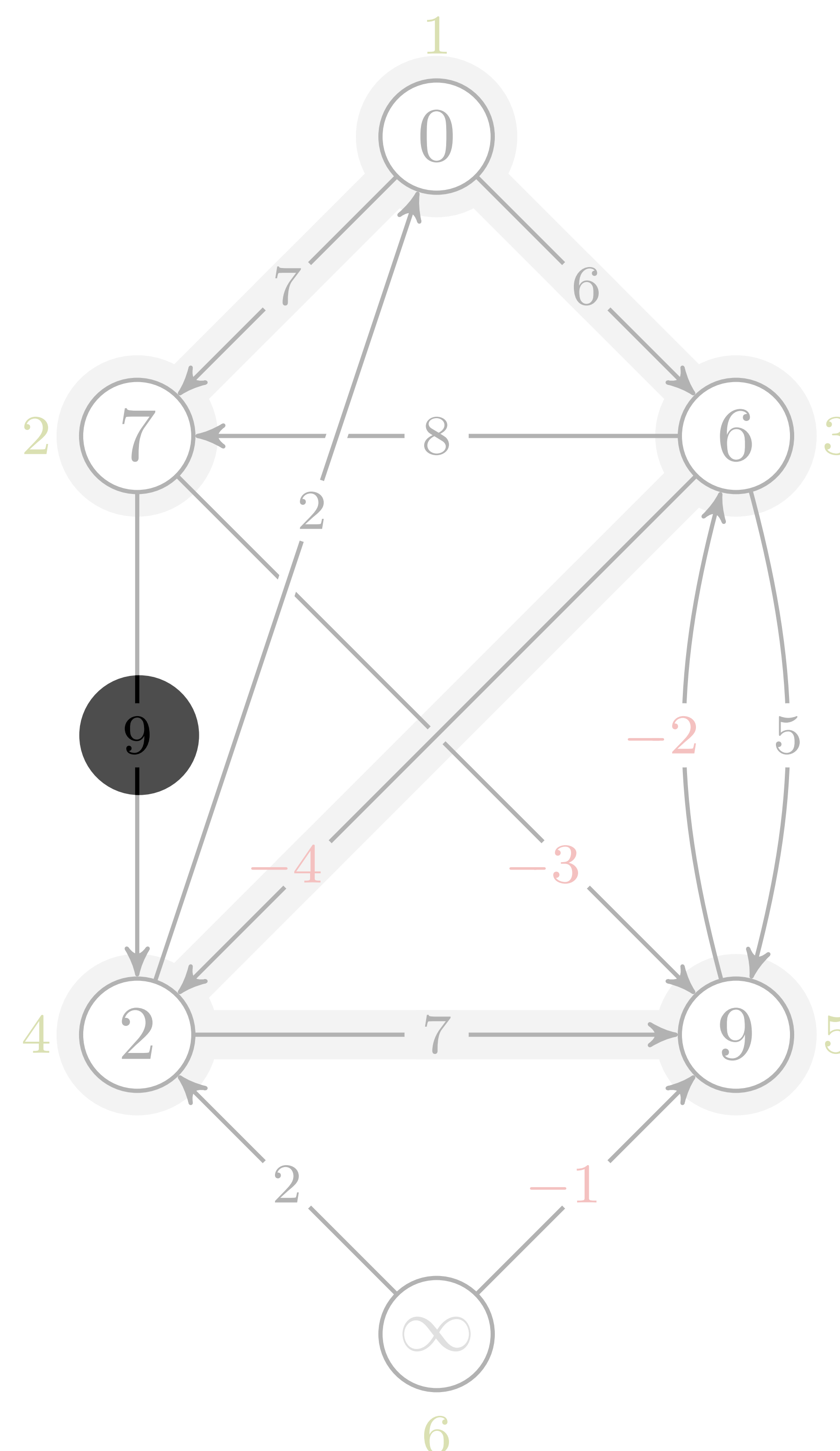


```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



$i, u, v \;=\; 1, \text{-}, \text{-}$

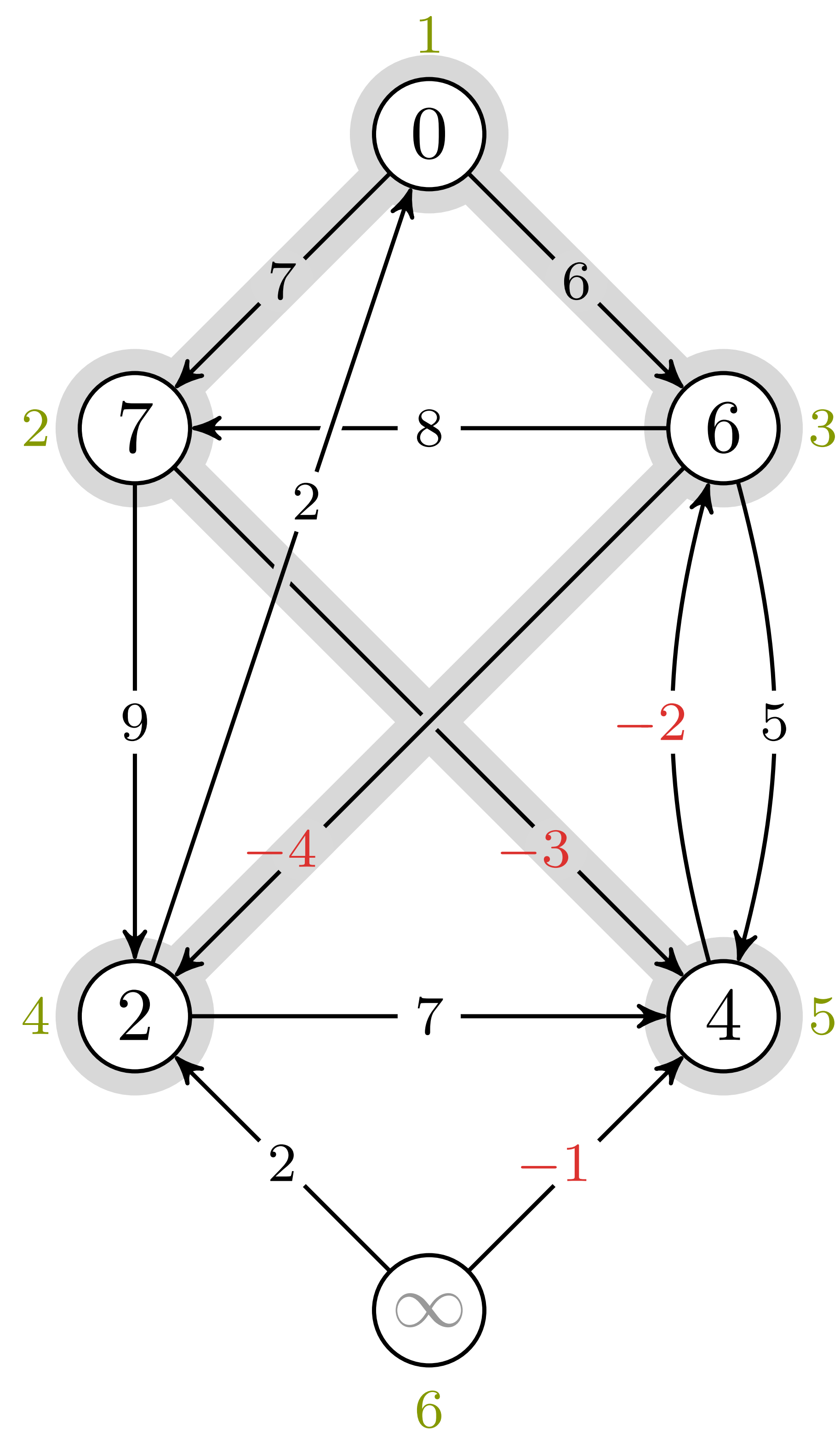
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



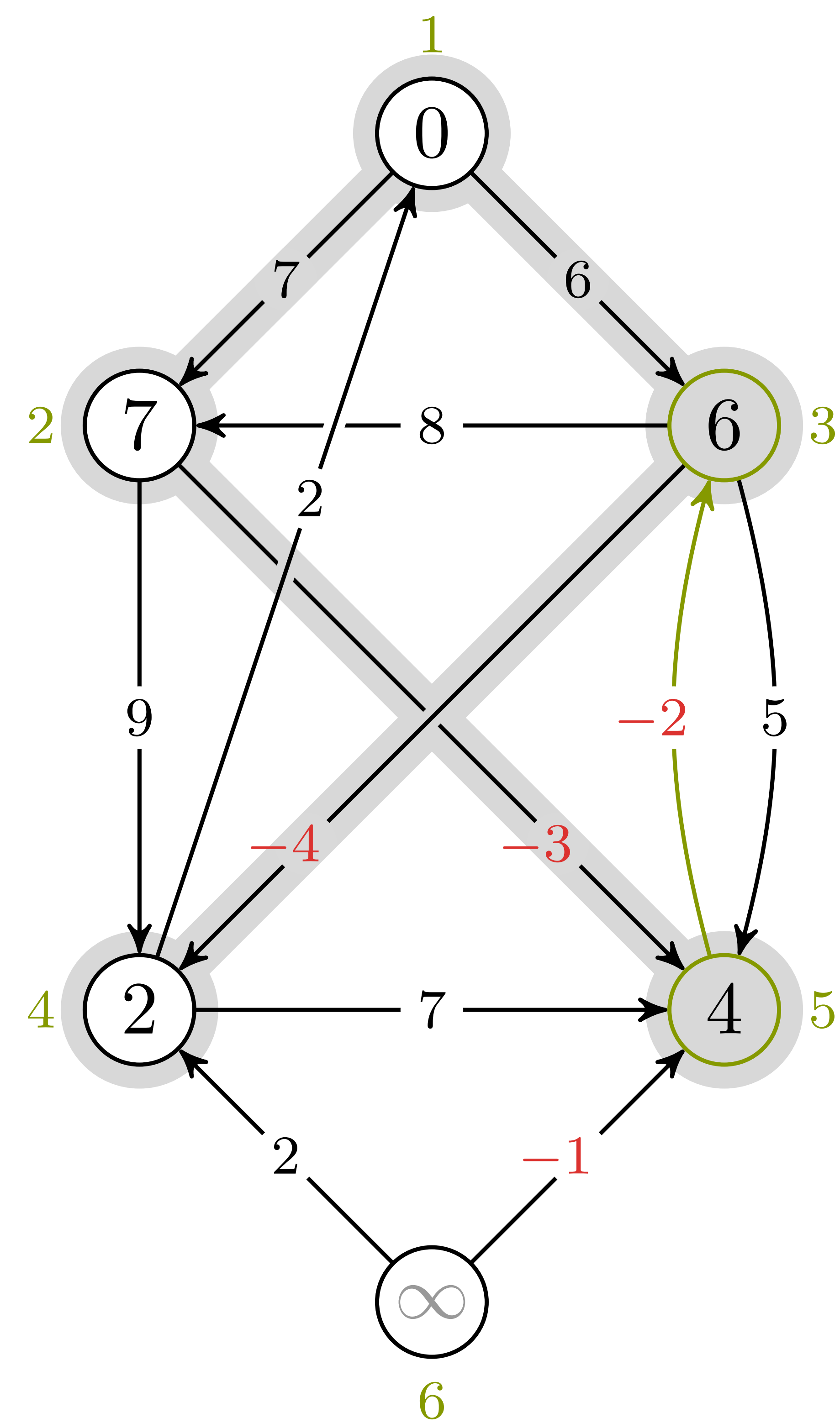
$i, u, v \ = \ 2, \text{-}, \text{-}$

```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

$i, u, v \;=\; 3, \text{-}, \text{-}$



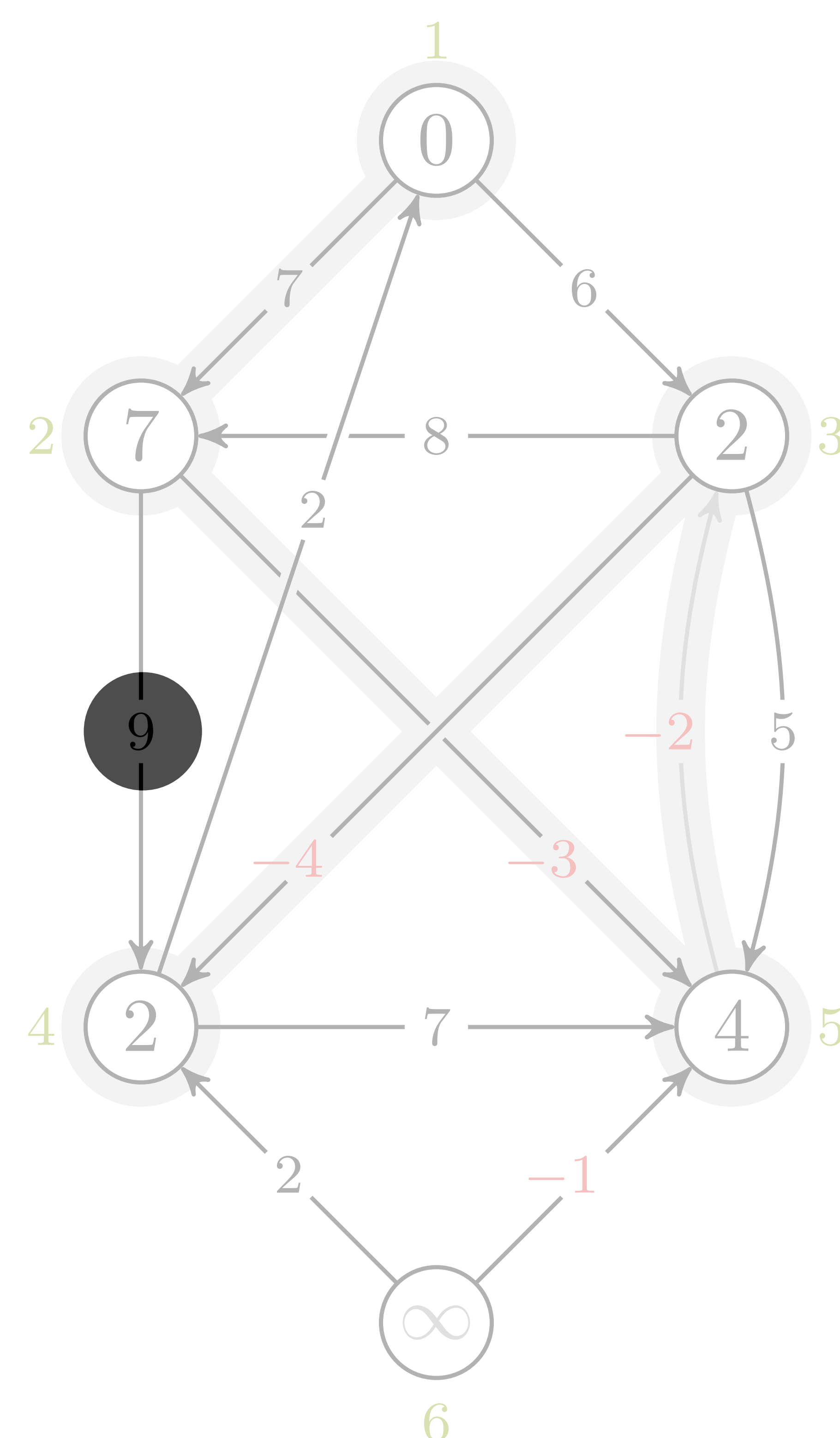
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



$i, u, v \;=\; 3, 5, 3$

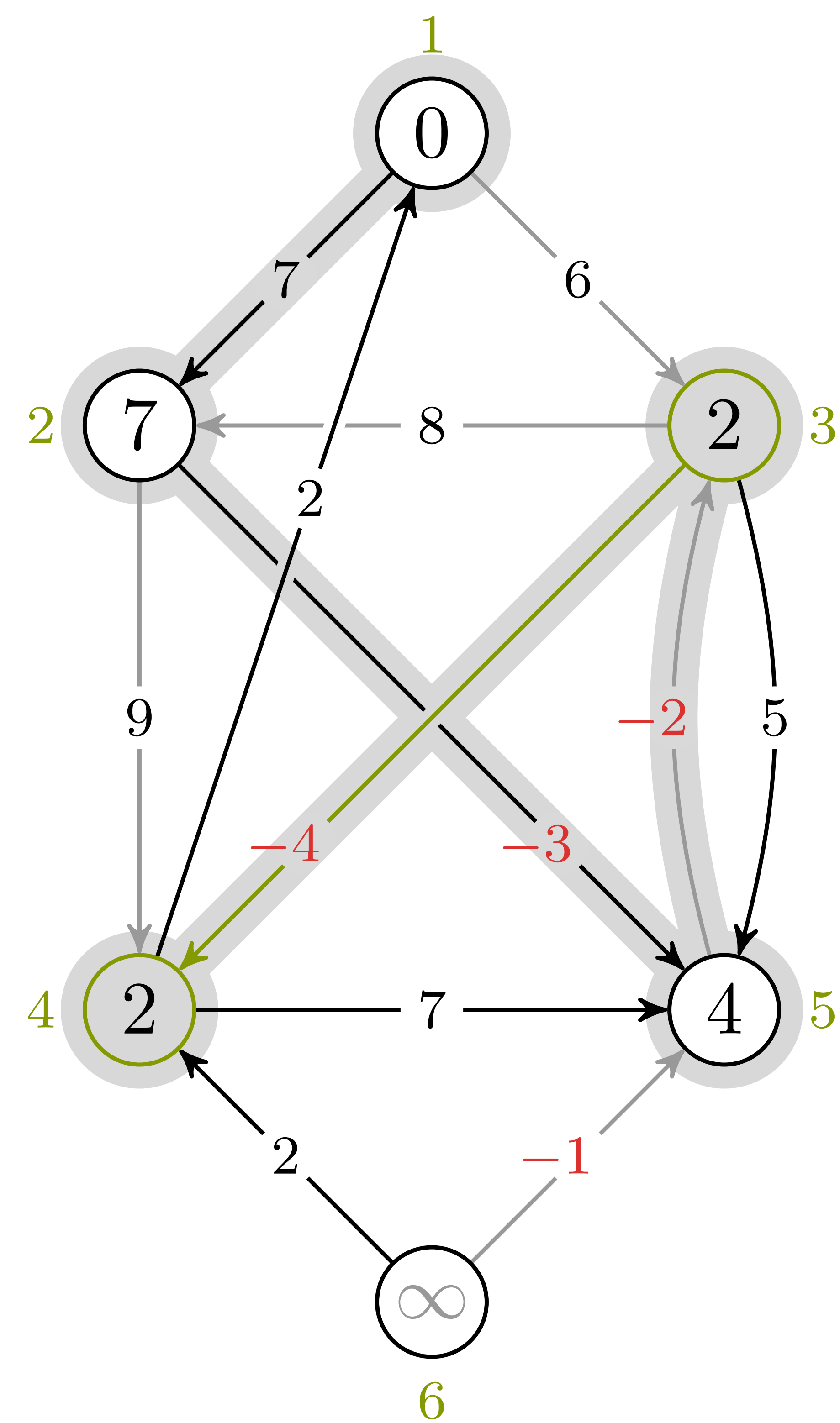
BELLMAN-FORD(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```



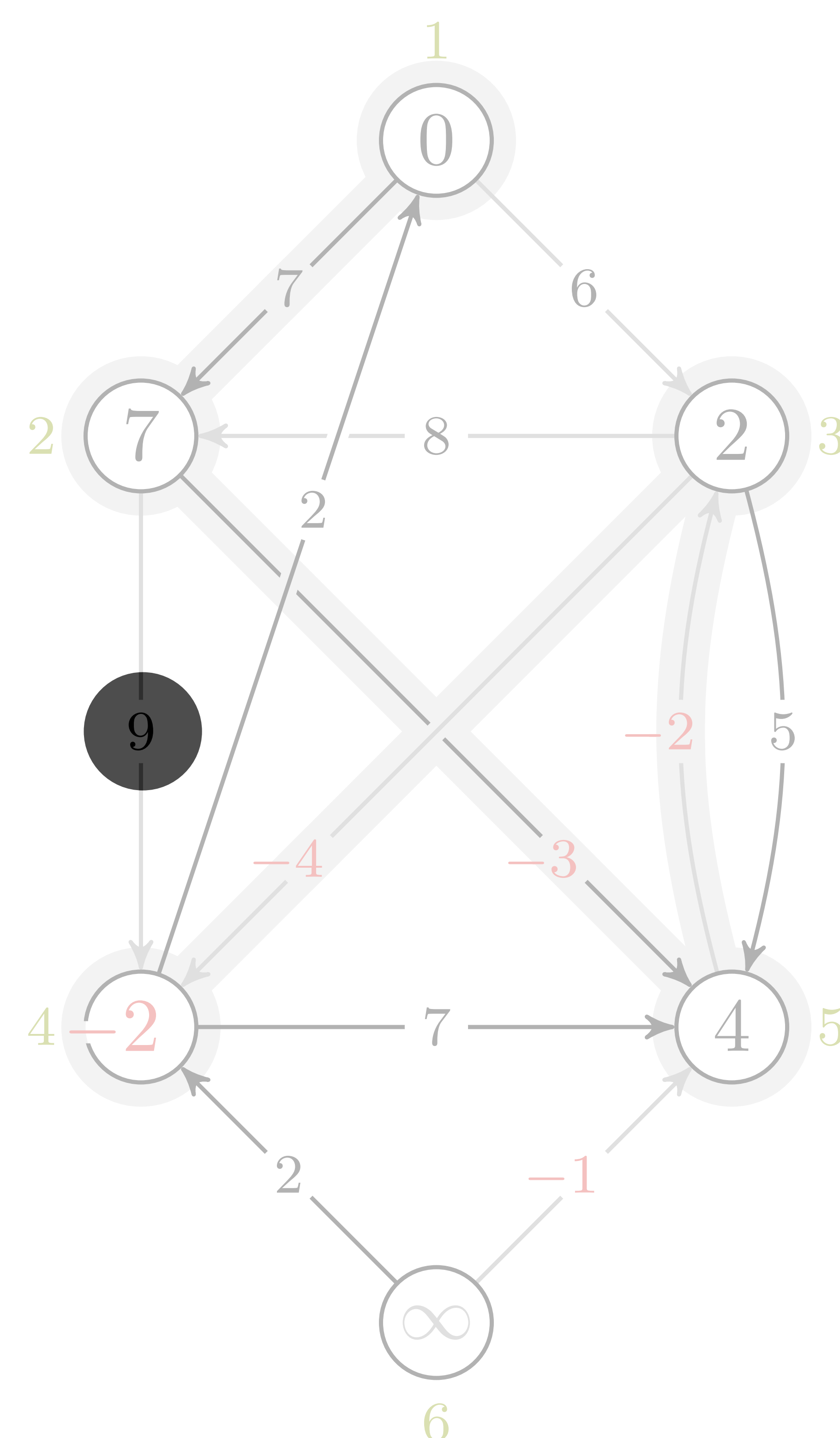
$i, u, v \;=\; 3, 5, 3$

```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



$i, u, v \ = \ 3, 3, 4$

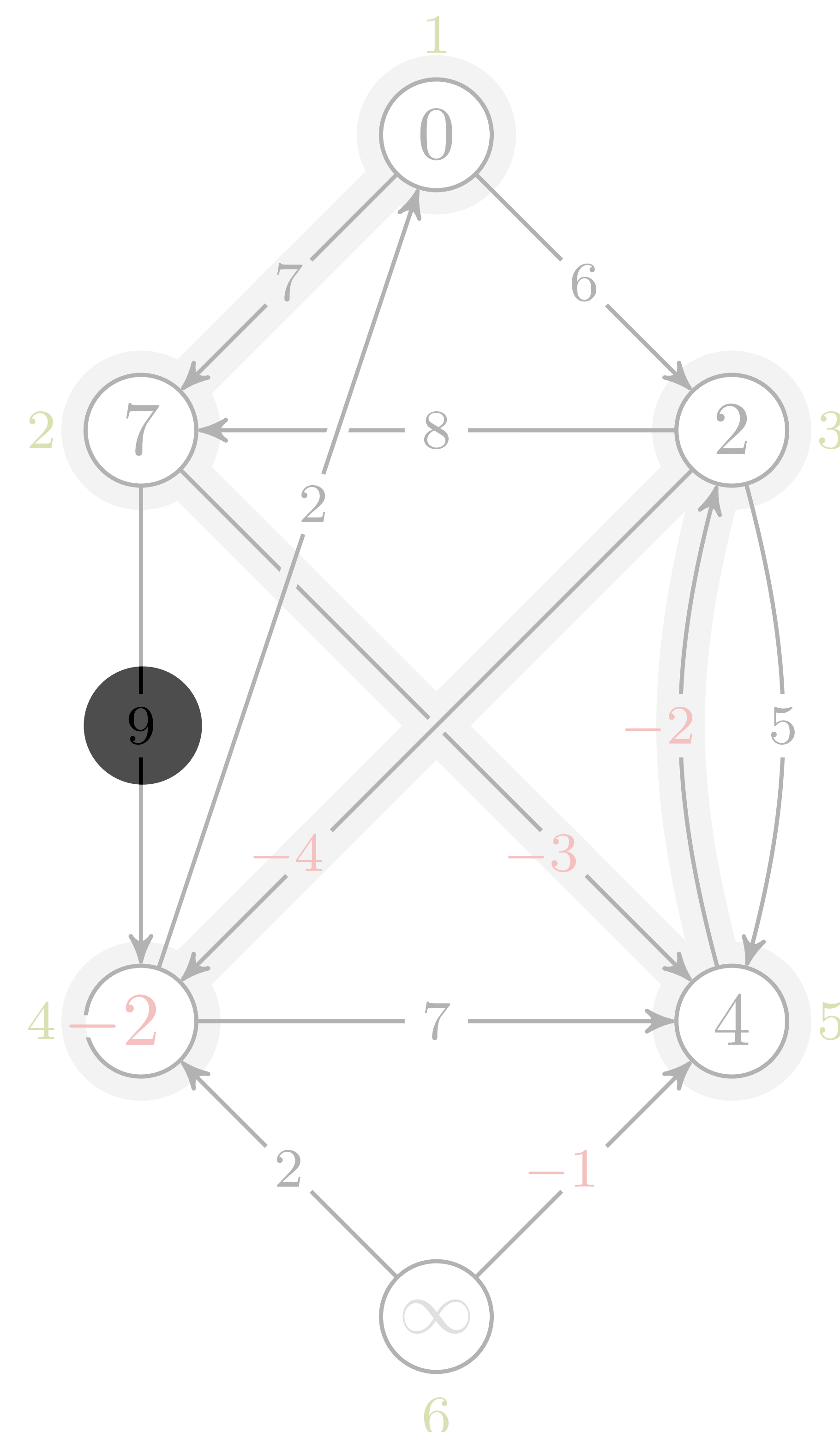
```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          Relax(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return false
  return true
```



$i, u, v \;=\; 3, 3, 4$

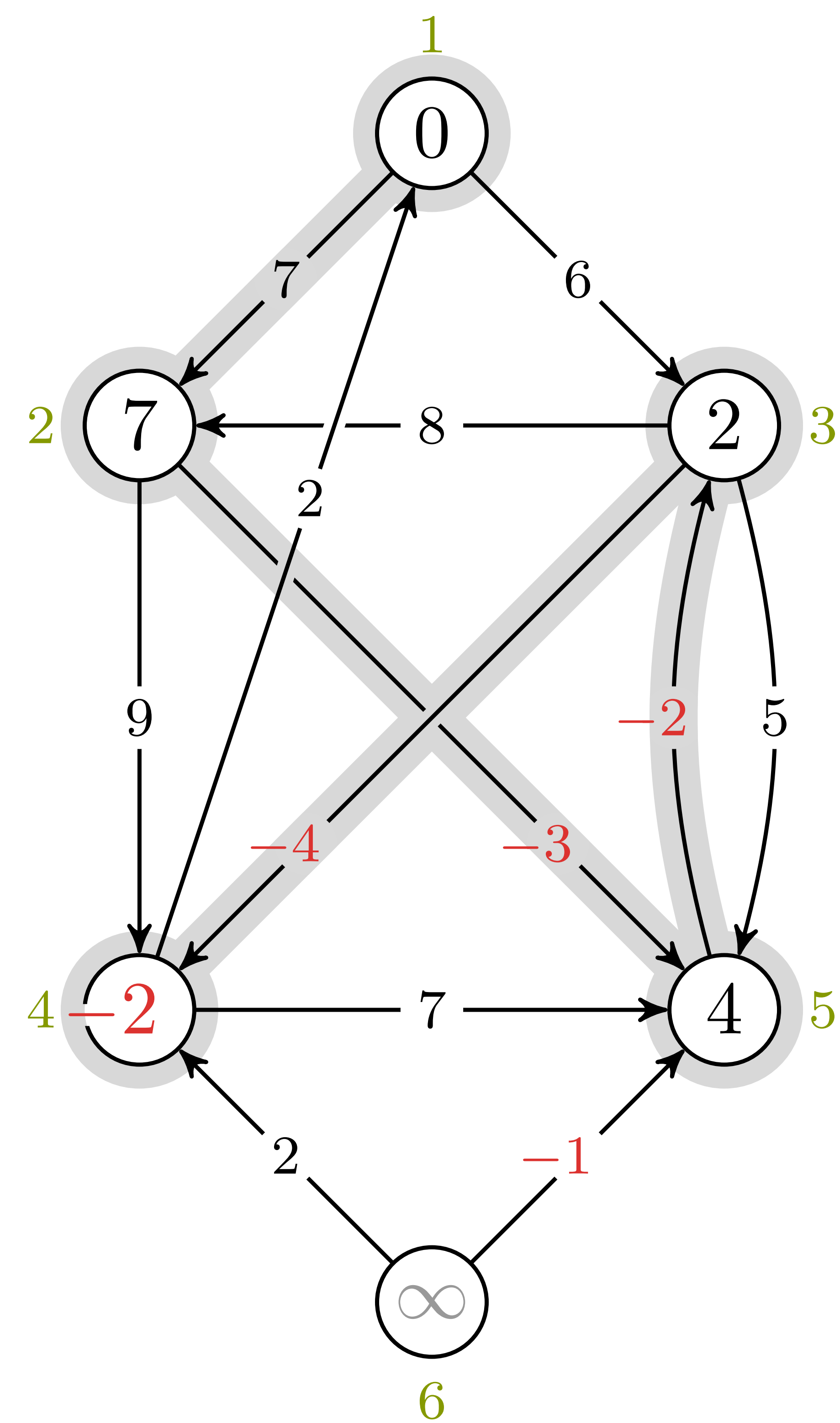
```
Bellman-Ford(G, w, s)
Initialize-Single-Source(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        Relax(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return false
return true
```

$i, u, v \;=\; 4, \text{-}, \text{-}$



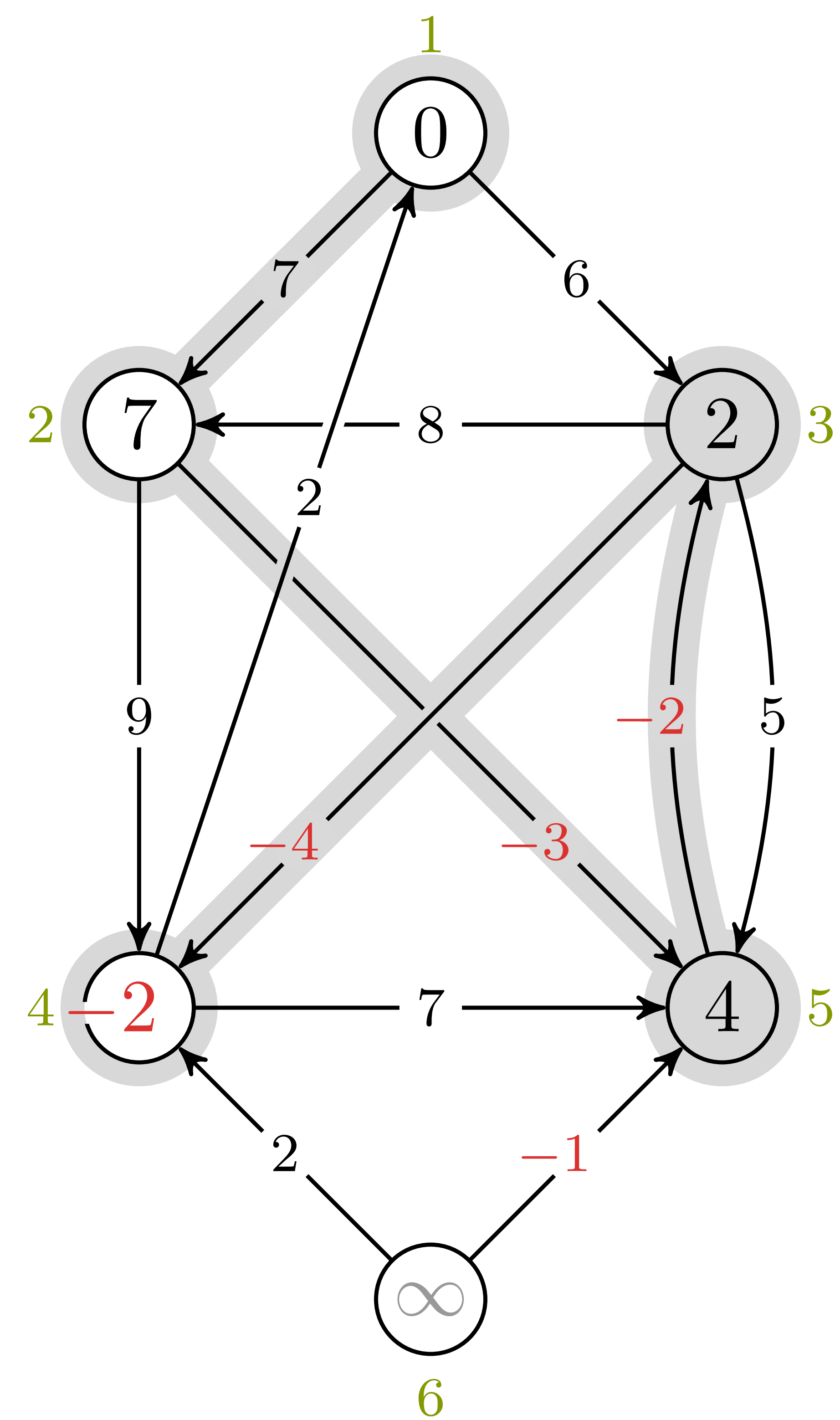
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

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



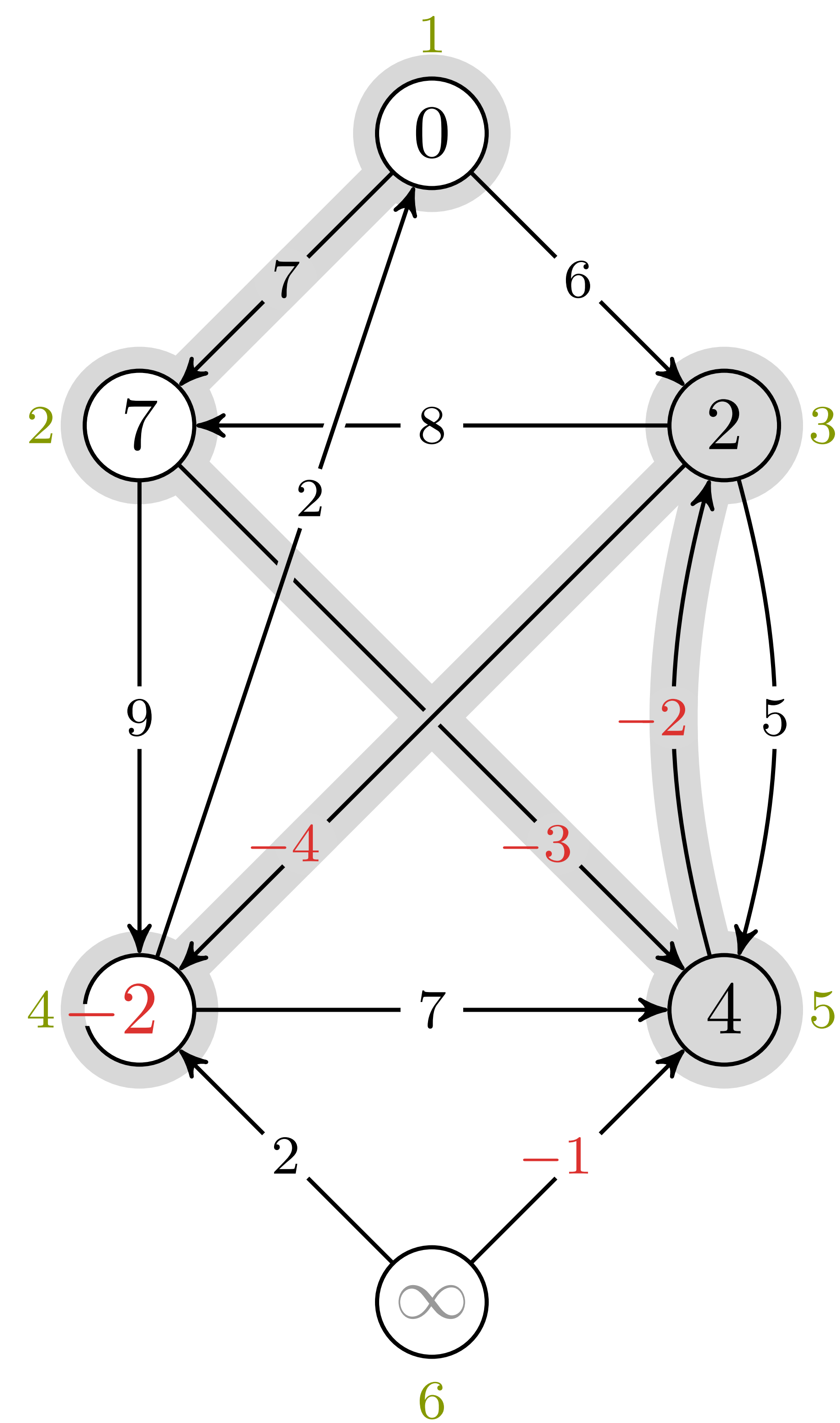
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

$i, u, v \ = \ \text{-}, 5, 3$

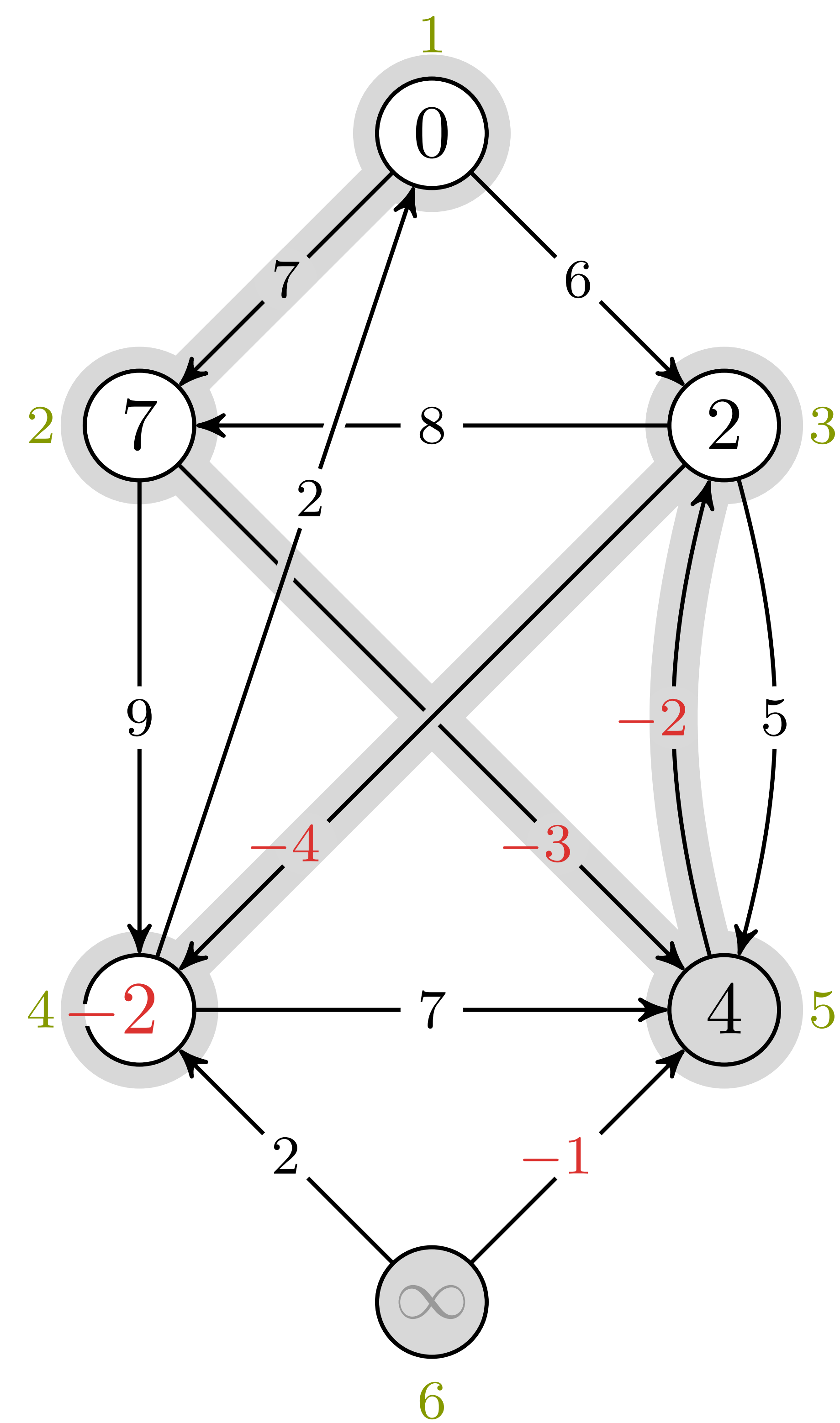


```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| - 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

$i, u, v \;=\; \text{-}, 5, 3$



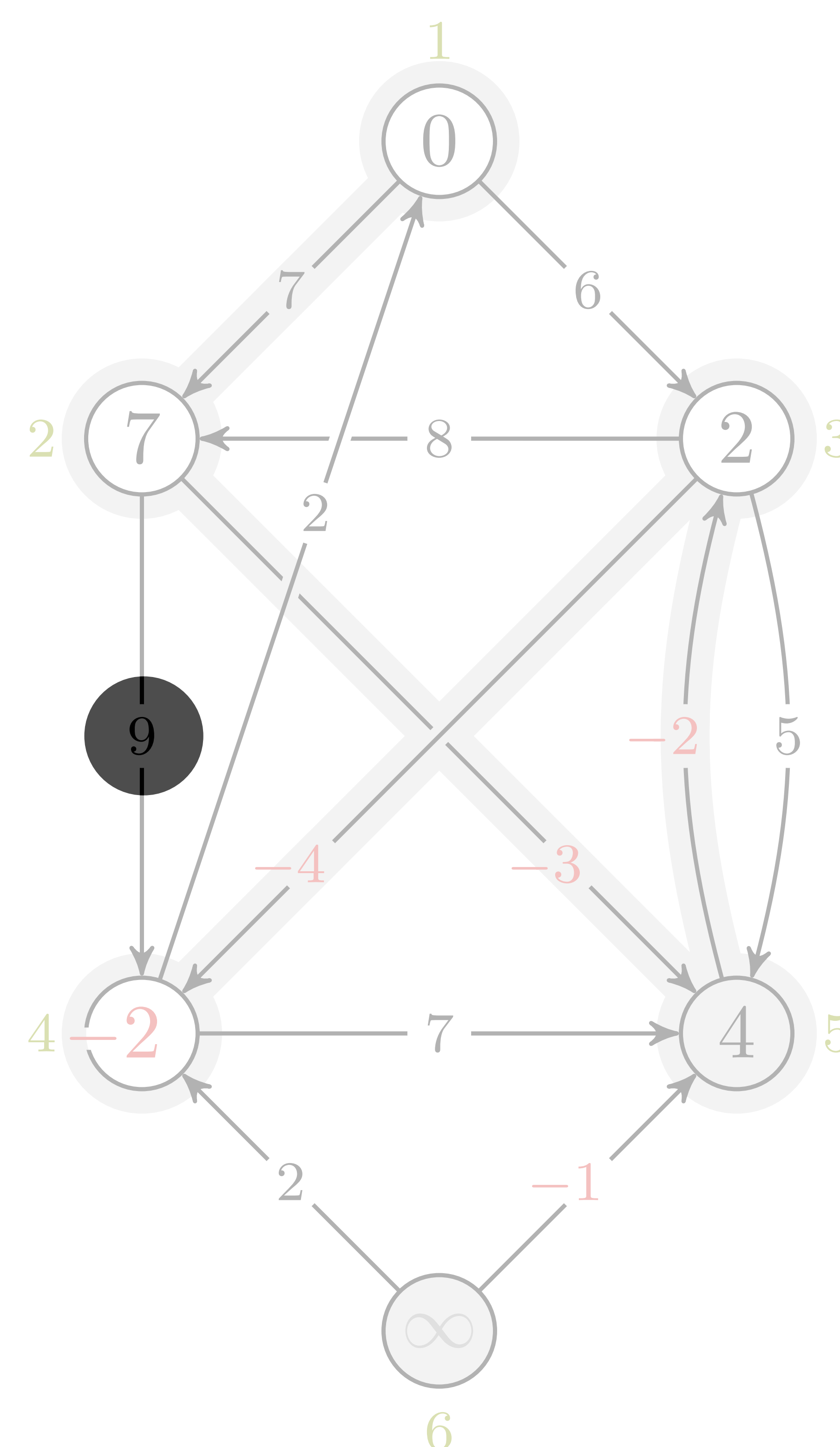
```
Bellman-Ford(G, w, s)
Initialize-Single-Source(G, s)
for i = 1 to |G.V| − 1
        for each edge (u, v) ∈ G.E
                Relax(u, v, w)
for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
                return false
return true
```



$i, u, v \ = \ \text{-}, 6, 5$

```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

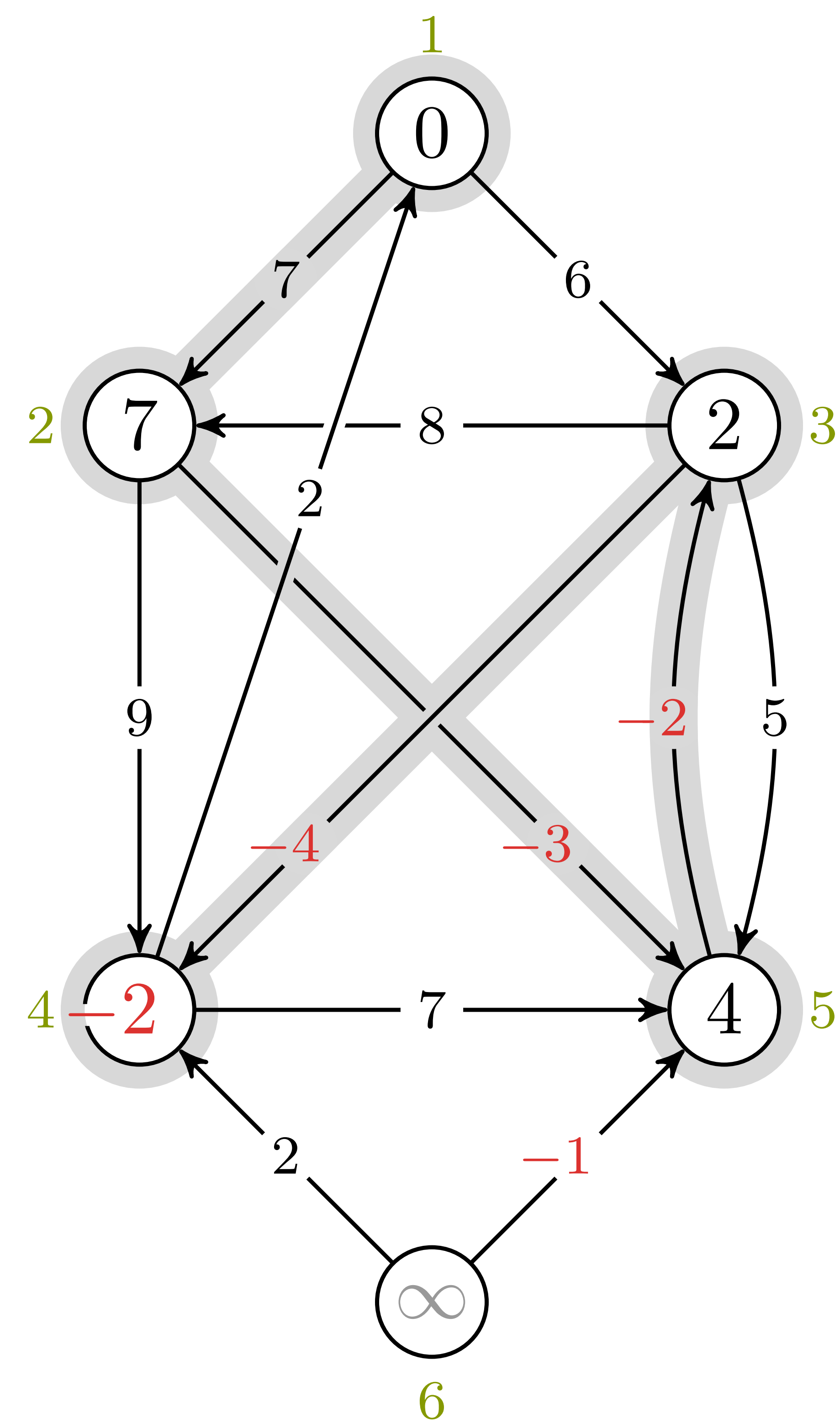
$i, u, v \;=\; \text{-}, 6, 5$



BELLMAN-FORD(G, w, s)

```
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| - 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

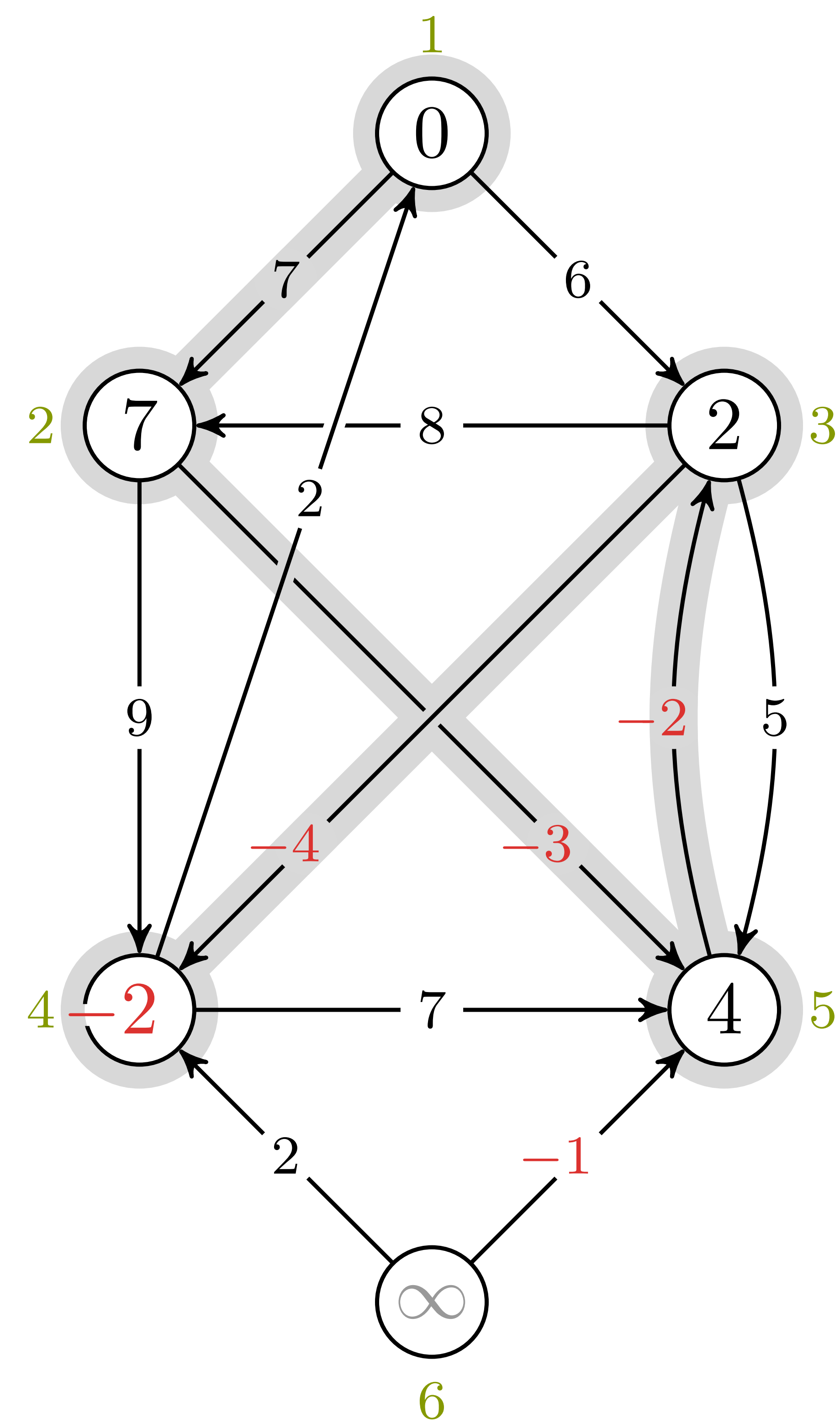
$i, u, v = -, -, -$



```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

→ TRUE

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

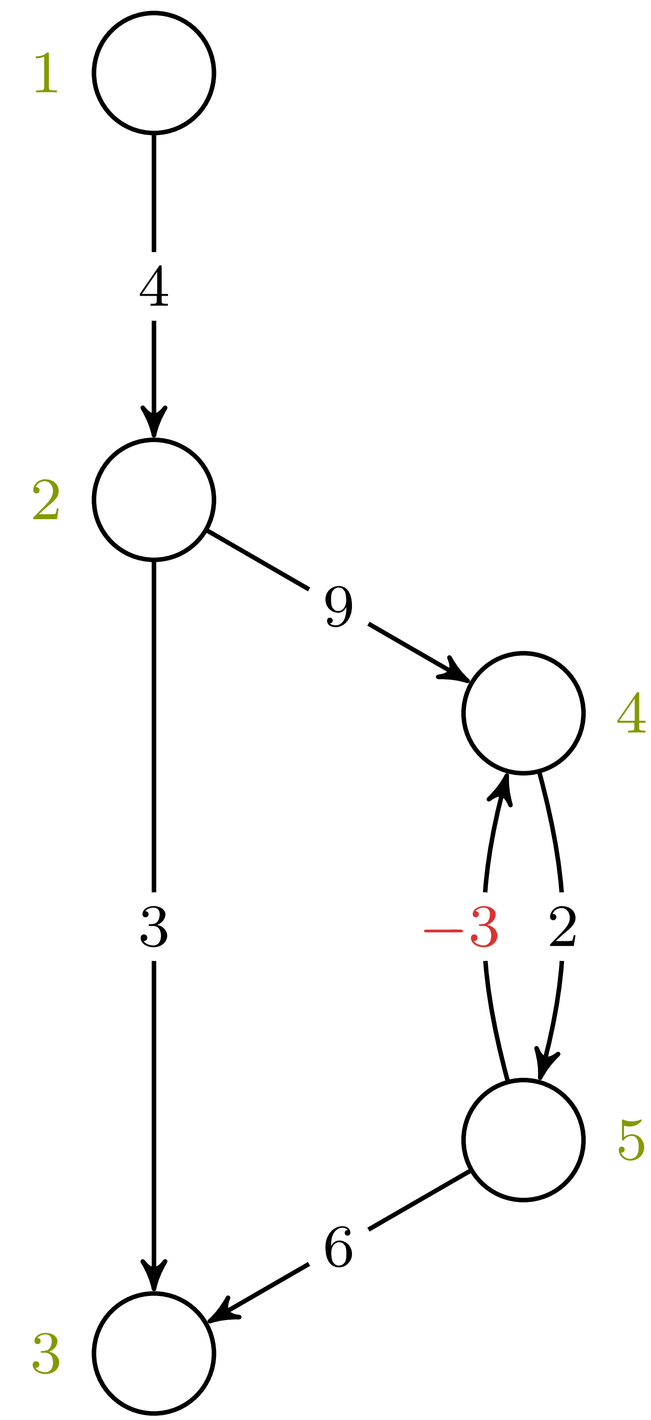


Negative sykler skaper trøbbel … det finnes fortsatt en kortest enkel sti, men vi klarer ikke finne den med denne fremgangsmåten.

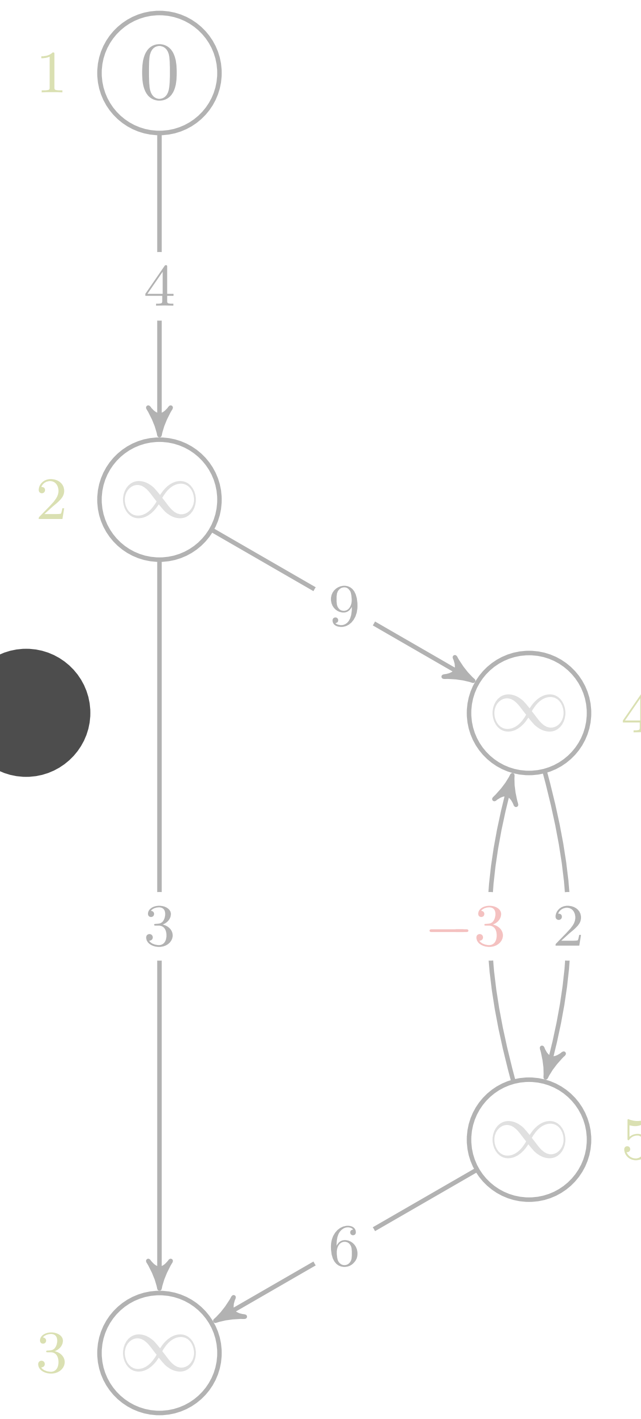
Bellman-Ford(G, w, s)

```
Initialize-Single-Source(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        Relax(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

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



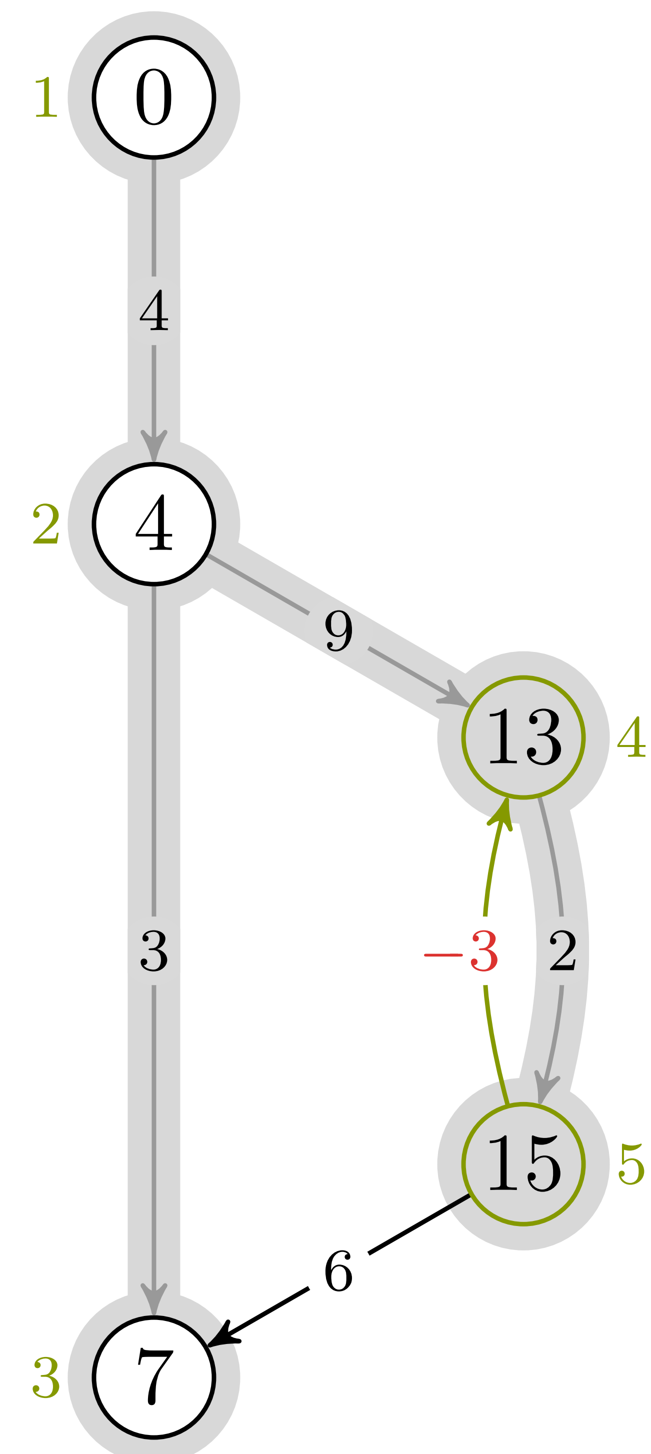
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



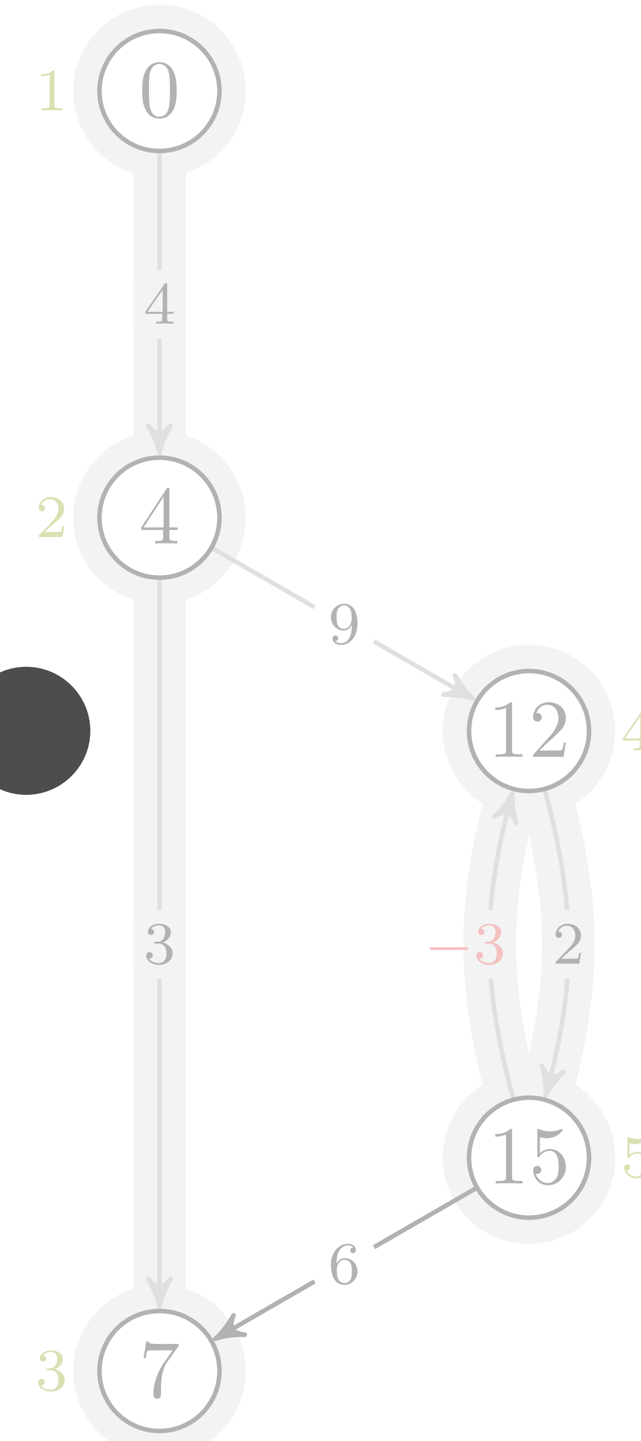
$i, u, v \;=\; \text{-}, \text{-}, \text{-}$

```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

$i, u, v \;=\; 2, 5, 4$

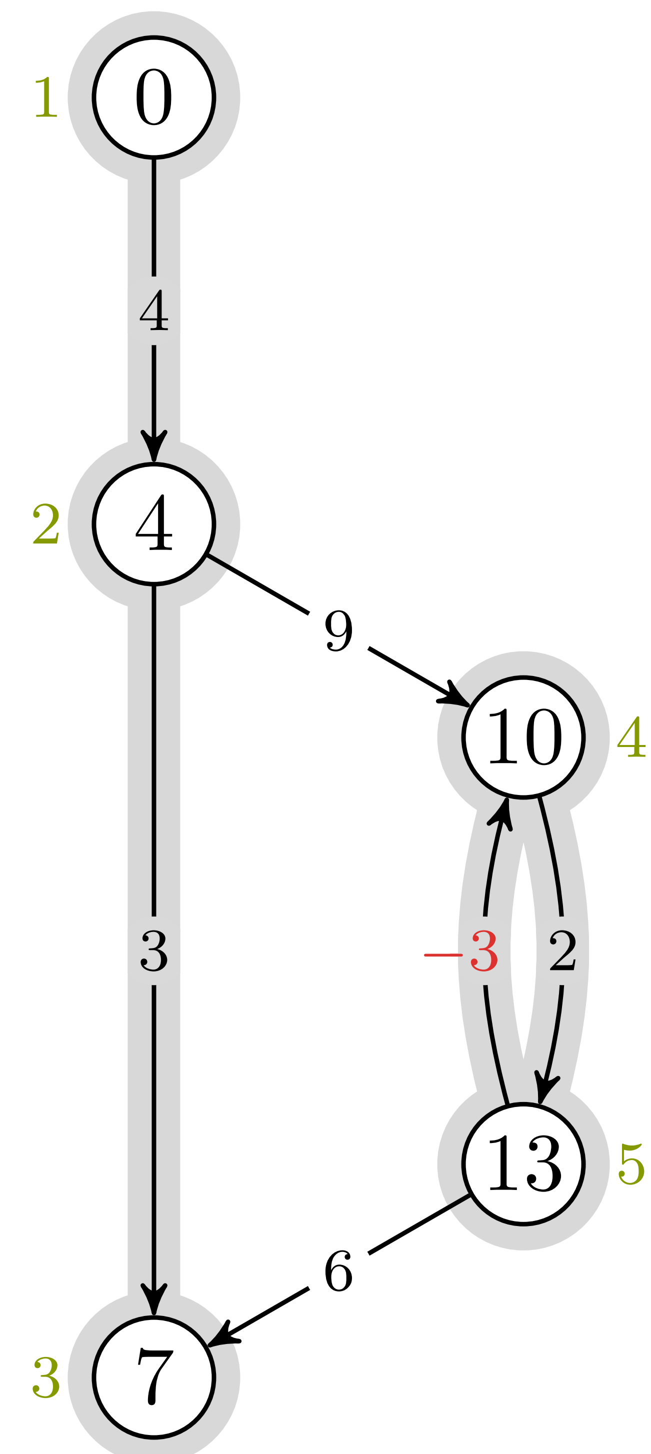


```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          Relax(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return false
  return true
```



$i, u, v \ = \ 2, 5, 4$

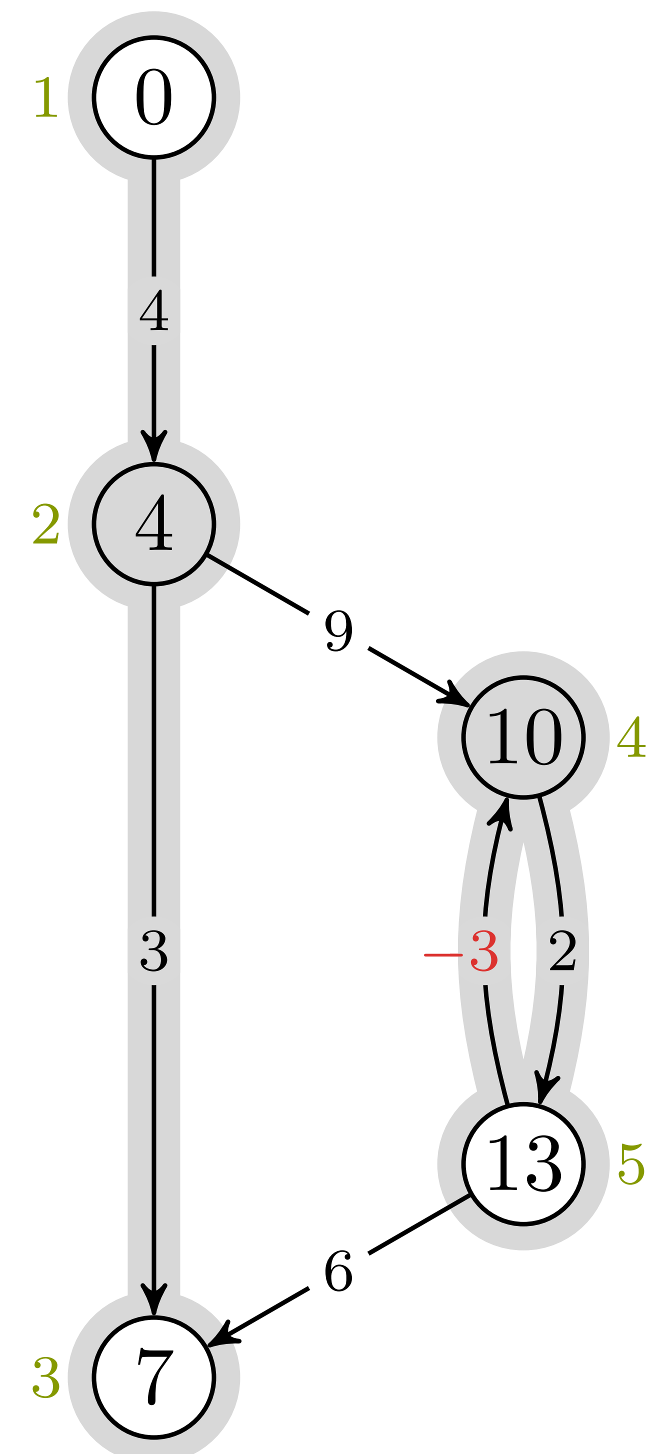
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



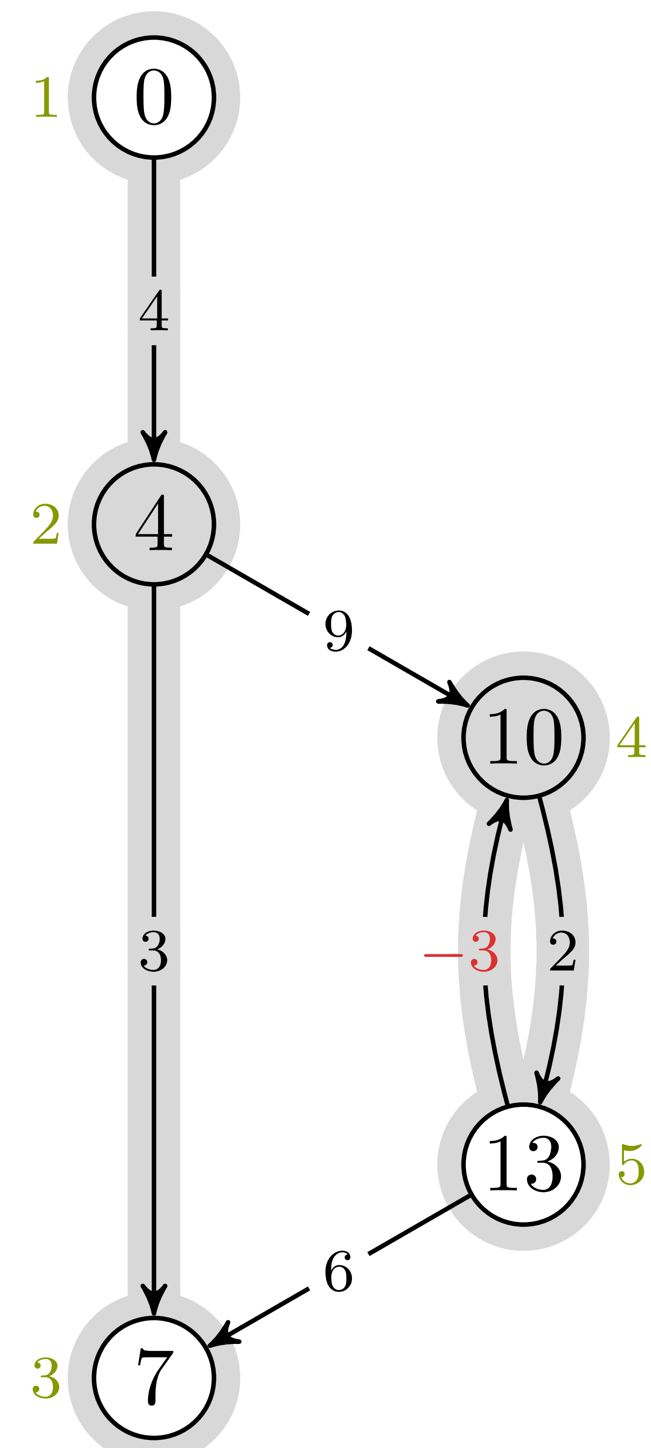
$i, u, v \;=\; \text{-}, \text{-}, \text{-}$

Bellman-Ford(G, w, s)
1 Initialize-Single-Source(G, s)
2 **for** $i = 1$ **to** $|G.V| - 1$
3 **for** each edge $(u, v) \in G.E$
4 Relax(u, v, w)
5 **for** each edge $(u, v) \in G.E$
6 **if** $v.d > u.d + w(u, v)$
7 **return** FALSE
8 **return** TRUE

$i, u, v = -, 2, 4$



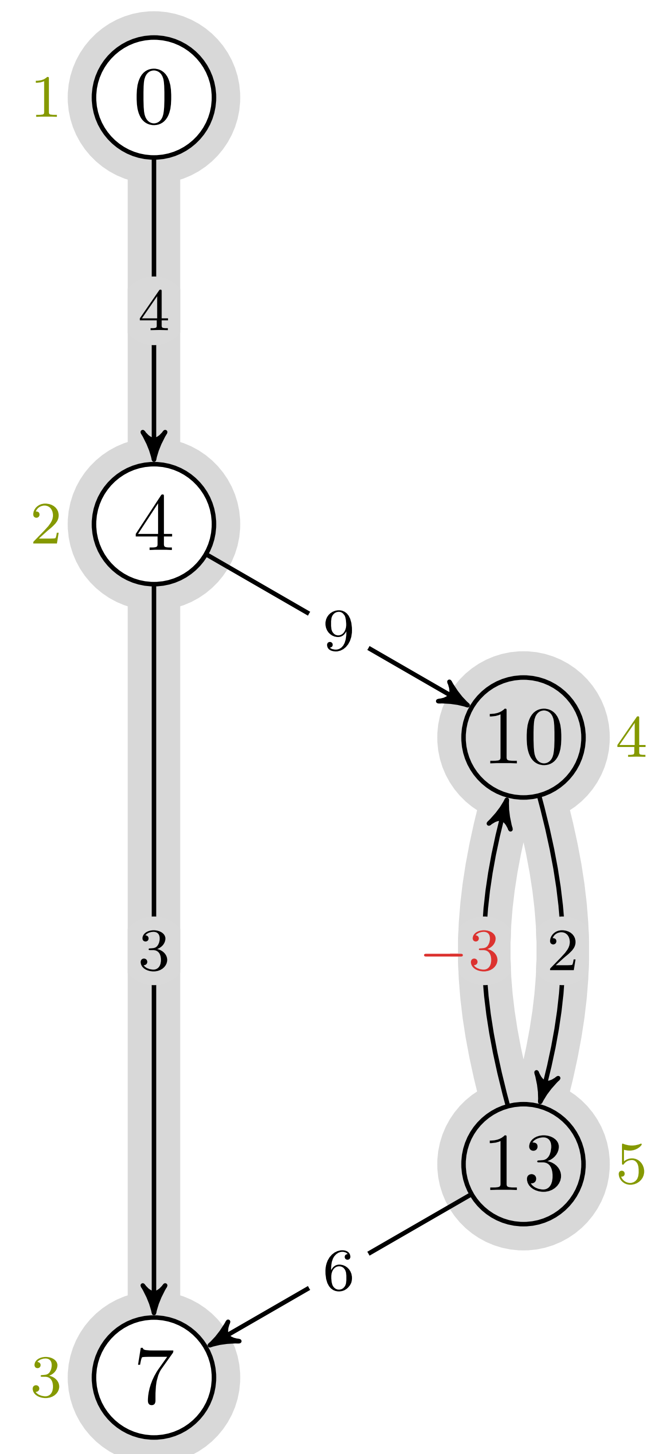
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



$i, u, v \;=\; \text{-}, 2, 4$

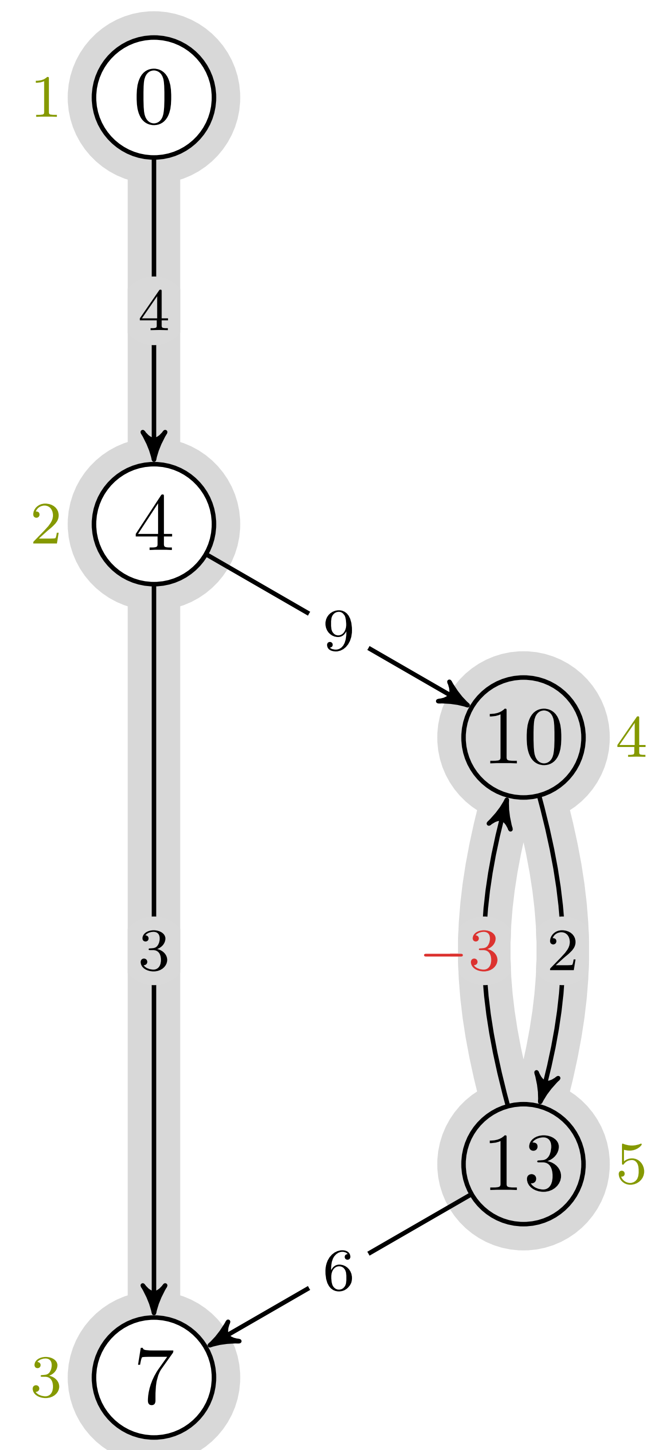
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

$i, u, v \;=\; \text{-}, 4, 5$



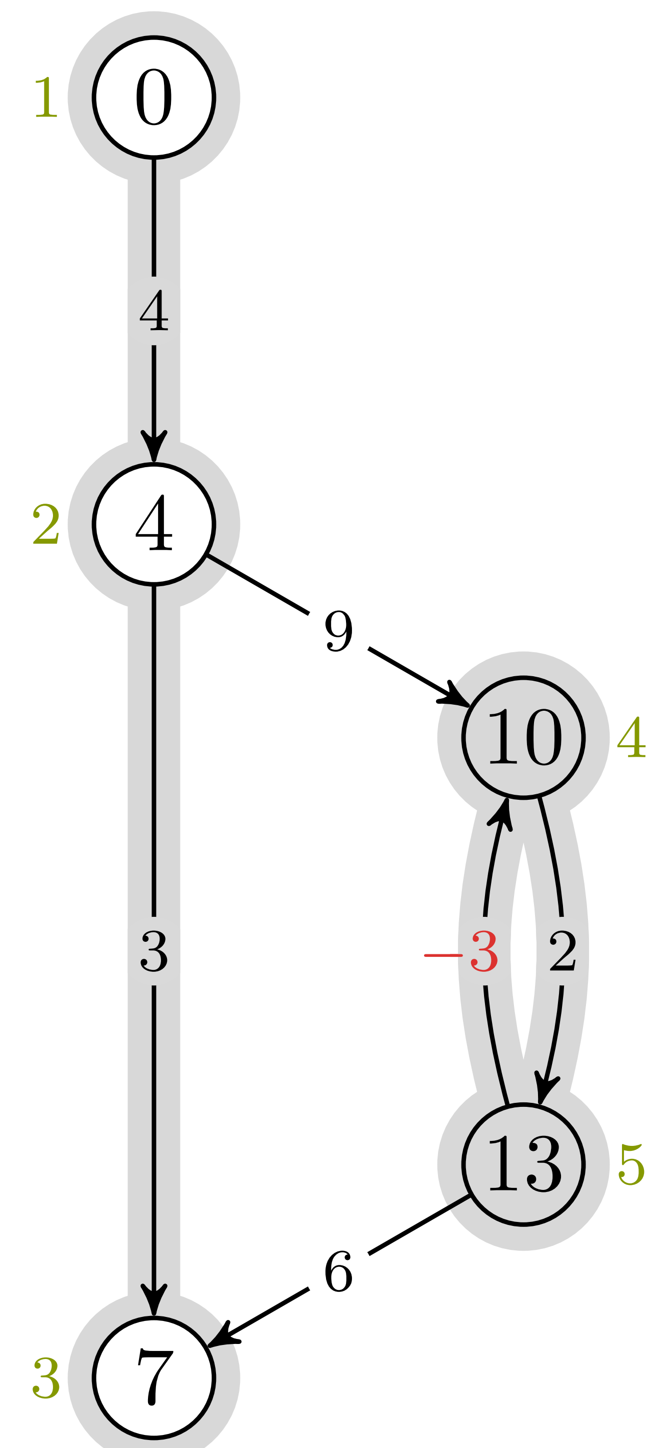
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

$i, u, v \;=\; \text{-}, 4, 5$



```
Bellman-Ford(G, w, s)
Initialize-Single-Source(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        Relax(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

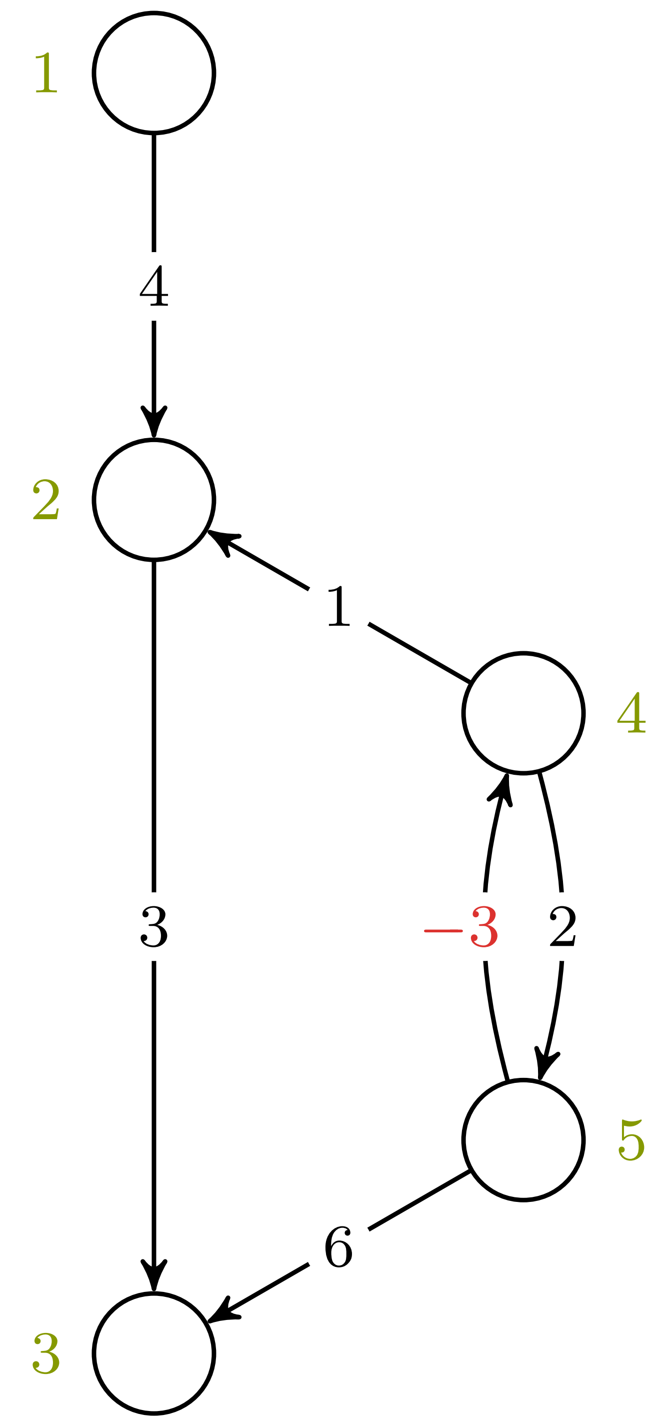
→ FALSE



i, u, v = -, -, -

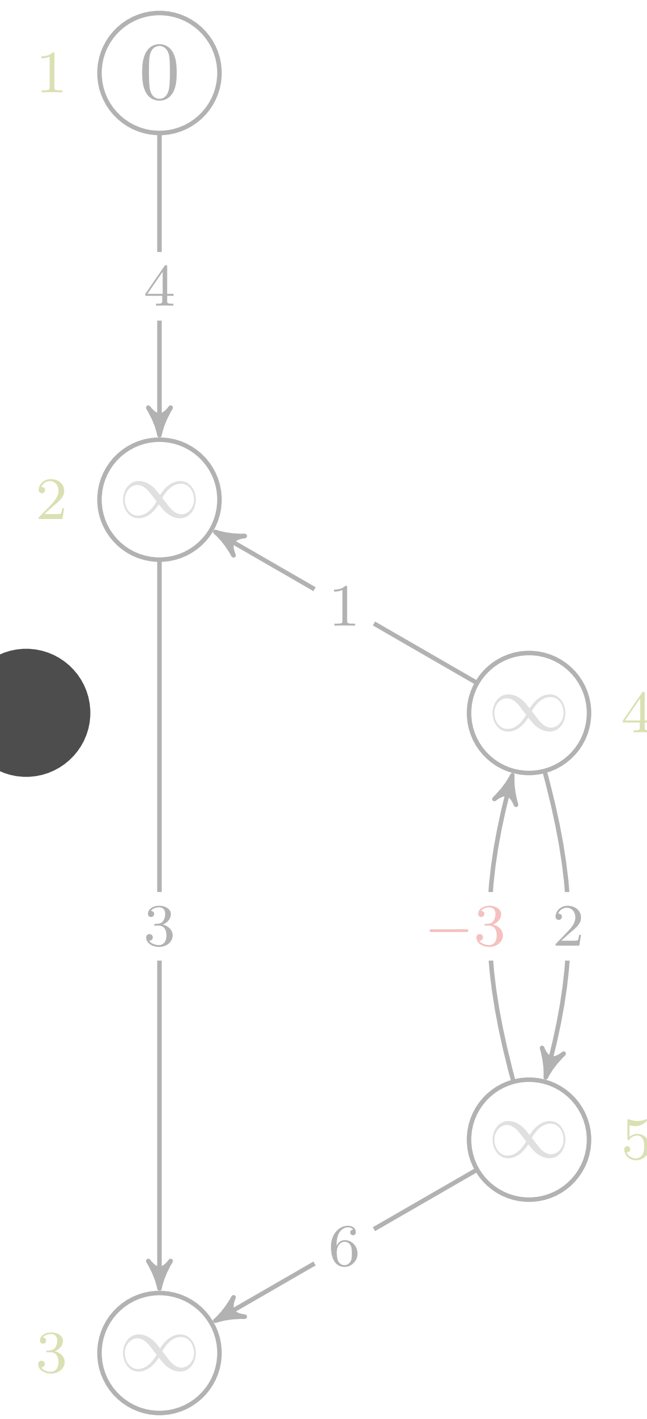
Merk at om ingen stier fra startnoden når frem til den negative sykelen, så skaper den *ikke* problemer, og løsningen vår er gyldig.

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```



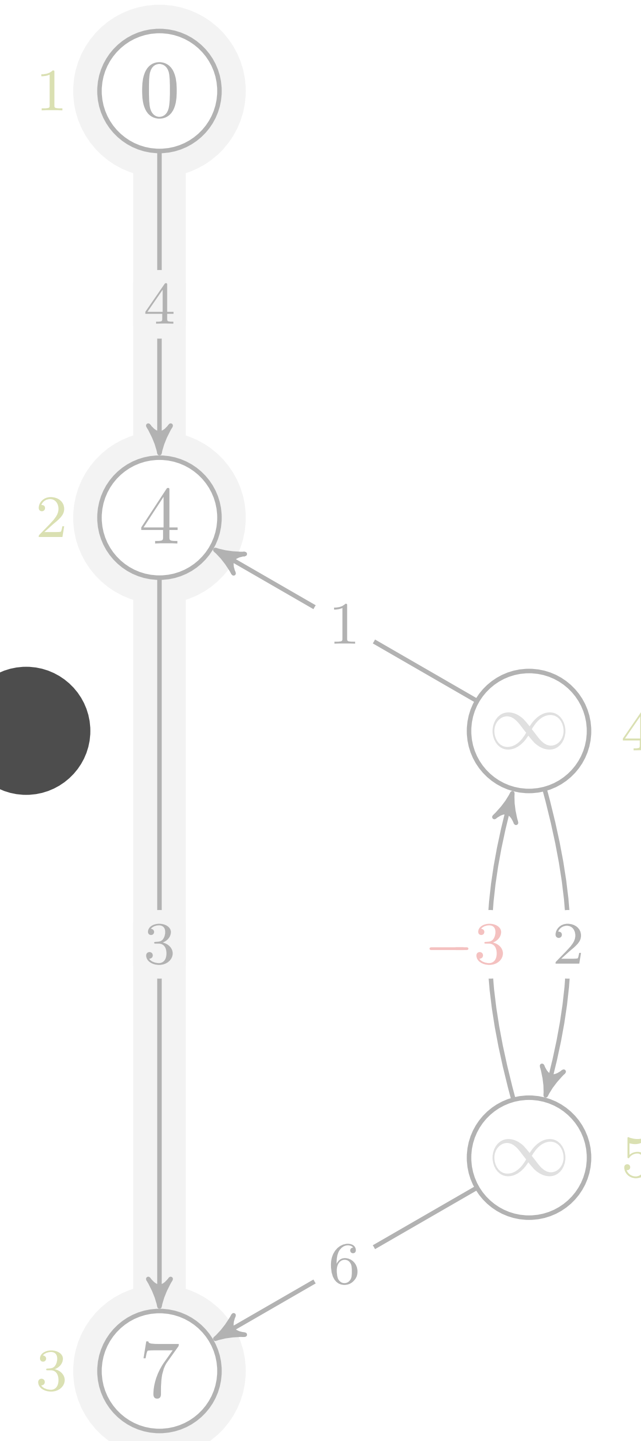
$i, u, v \; = \;$ -, -, -

```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
        for each edge (u, v) ∈ G.E
              Relax(u, v, w)
  for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
              return false
  return true
```



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

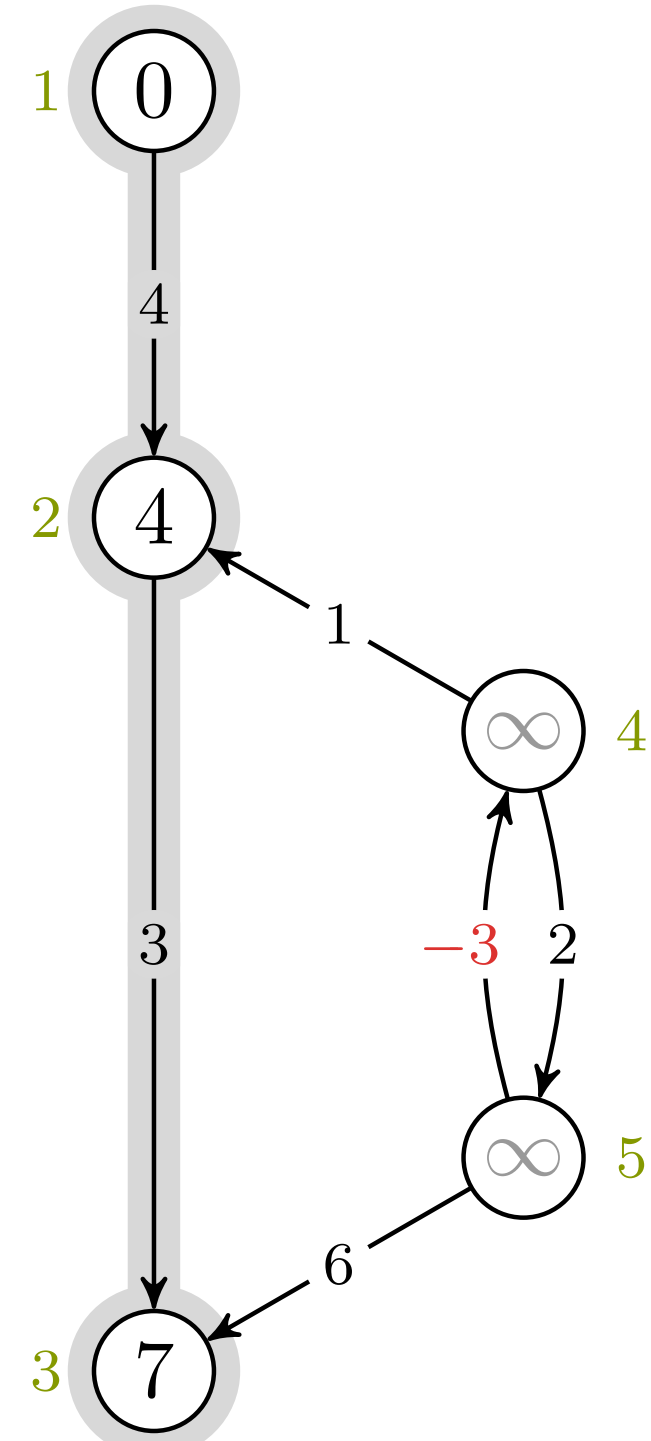
```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```



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

```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          Relax(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return false
  return true
```

→ TRUE



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

Bellman-Ford ›

Kjøretid

Se også oppg. 24-1

Operasjon	Antall	Kjøretid

Operasjon	Antall	Kjøretid
Initialisering		

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
Relax		

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
Relax	$V-1$	

(Altså slakking av alle kantene)

Operasjon	Antall	Kjøretid
Initialisering	1	$\Theta(V)$
Relax	$V - 1$	$\Theta(E)$

Operasjon	Antall	Kjøretid
Initialisering	1	$\Theta(V)$
Relax	$V-1$	$\Theta(E)$
Relax		

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
Relax	$V - 1$	$\Theta(E)$
Relax	1	

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
RELAX	$V - 1$	$\Theta(E)$
RELAX	1	$O(E)$

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
RELAX	$V - 1$	$\Theta(E)$
RELAX	1	$O(E)$

Totalt: $\Theta(VE)$

Her oppgir boka bare den øvre grensen, O(VE).

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
Relax	$V-1$	$\Theta(E)$
Relax	1	$O(E)$

Totalt: $\Theta(VE)$

Stopp når ingenting endres? Fortsatt $O(VE)$

Om et estimat endres, så var tidligere slakking fra noden bortkastet.

Om et estimat endres, så var tidligere slakking fra noden bortkastet.

Konklusjon:

Slakk kanter fra v når $v.d$ ikke kan forbedres.

Om et estimat endres, så var tidligere slakking fra noden bortkastet.

Konklusjon:

Slakk kanter fra v når $v.d$ ikke kan forbedres.

Og hvis $\delta(s, v) < v.d$ så kan $u.d$ forbedres (via korteste vei)

Om et estimat endres, så var tidligere slakking fra noden bortkastet.

Konklusjon:

Slakk kanter fra v når $v.d$ ikke kan forbedres.

Men hvordan vet vi når $v.d$ ikke kan forbedres?

Det vi startet med!

Strategi 1 av 2:

Slakk kanter ut fra noder i topologisk sortert rekkefølge.

Strategi 1 av 2:

Slakk kanter ut fra noder i topologisk sortert rekkefølge.*

* Krever en rettet asyklisk graf

Strategi 1 av 2:

Slakk kanter ut fra noder i topologisk sortert rekkefølge.*

Hvorfor blir det rett?

* Krever en rettet asyklisk graf

Strategi 1 av 2:

Slakk kanter ut fra noder i topologisk sortert rekkefølge.*

Hvorfor blir det rett?

* Krever en rettet asyklisk graf

(Implisitt induksjon: Anta at tidligere noder har fått rett estimat)

Strategi 1 av 2:

Slakk kanter ut fra noder i topologisk sortert rekkefølge.*

Hvorfor blir det rett?

Når alle inn-kanter er slakket kan ikke noden forbedres, og kan trygt velges som neste.

* Krever en rettet asyklisk graf

(Implisitt induksjon: Anta at tidligere noder har fått rett estimat)

Strategi 1 av 2:

Slakk kanter ut fra noder i topologisk sortert rekkefølge.*

Hvorfor blir det rett?

Når alle inn-kanter er slakket kan ikke noden forbedres, og kan trygt velges som neste.

* Krever en rettet asyklisk graf

Evt.: Alle stier, inkl. korteste, får kanter slakket i rekkefølge

Hva om vi vil ha sykler?

Hva om vi vil ha sykler?

Strategi 2 av 2:

Velg den gjenværende med lavest estimat.

Strategi 2 av 2:

Velg den gjenværende med lavest estimat.

Hvorfor blir det rett?

Strategi 2 av 2:

Velg den gjenværende med lavest estimat.

Hvorfor blir det rett?

(Nok en gang: Anta at tidligere noder har fått rett estimat)

Strategi 2 av 2:

Velg den gjenværende med lavest estimat.

Hvorfor blir det rett?

Gjenværende noder kan kun forbedres ved slakking fra andre gjenværende. Det laveste estimatet kan dermed ikke forbedres.

(Nok en gang: Anta at tidligere noder har fått rett estimat)

Strategi 2 av 2:

Velg den gjenværende med lavest estimat.

Hvorfor blir det rett?

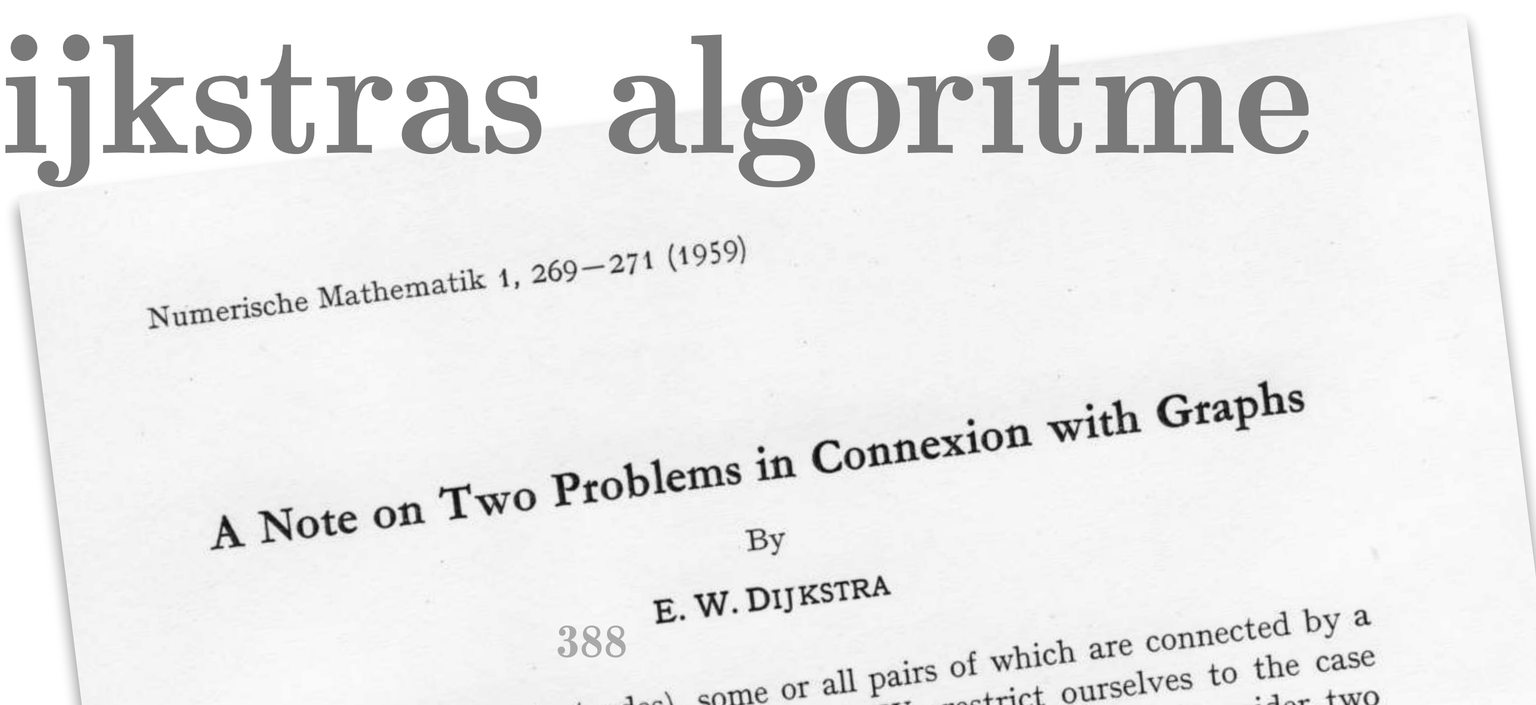
Gjenværende noder kan kun forbedres ved slakking fra andre gjenværende. Det laveste estimatet kan dermed ikke forbedres.*

* Stemmer ikke hvis vi har negative kanter!

(Nok en gang: Anta at tidligere noder har fått rett estimat)

5:5

Dijkstras algoritme


Numerische Mathematik 1, 269—271 (1959)

A Note on Two Problems in Connexion with Graphs

By

E. W. Dijkstra

some or all pairs of which are connected by a

restrict ourselves to the case

$\textsc{Dijkstra}(G, w, s)$

G	graf
w	vekting
s	startnode

Slakk ut-kanter fra den gjenværende med lavest estimat

DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)

G graf
w vekting
s startnode

DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $\mathrm{S} = \emptyset$

G	graf
w	vekting
s	startnode
S	ferdige

Ferdige: Vi har slakket ut fra dem og de vil aldri endre seg igjen

DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S = ∅
3 Q = G.V

G	graf
w	vekting
s	startnode
S	ferdige
Q	pri-kø

Prioritetskø som i PRIM, men her prioritert etter $v.d$

DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S = \emptyset$
3 $Q = G.V$
4 **while** $Q \neq \emptyset$

G	graf
w	vekting
s	startnode
S	ferdige
Q	pri-kø

Så lenge det er noen vi ikke har avstanden til . . .

DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN(Q)

G	graf
w	vekting
s	startnode
S	ferdige
Q	pri-kø
u	fra-node

Ingen negative kanter $\implies$ $u.d$ vil aldri endres mer!

Dijkstra(G, w, s)

Initialize-Single-Source(G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u =$ Extract-Min(Q)
    $S = S \cup \{u\}$

G	graf
w	vekting
s	startnode
S	ferdige
Q	pri-kø
u	fra-node

Dijkstra(G, w, s)
Initialize-Single-Source(G, s)
S = $\emptyset$
Q = G.V
while Q $\neq \emptyset$
    u = Extract-Min(Q)
    S = S $\cup \{u\}$
    for each vertex $v \in$ G.$Adj[u]$

G graf
w vekting
s startnode
S ferdige
Q pri-kø
u fra-node
v til-node

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

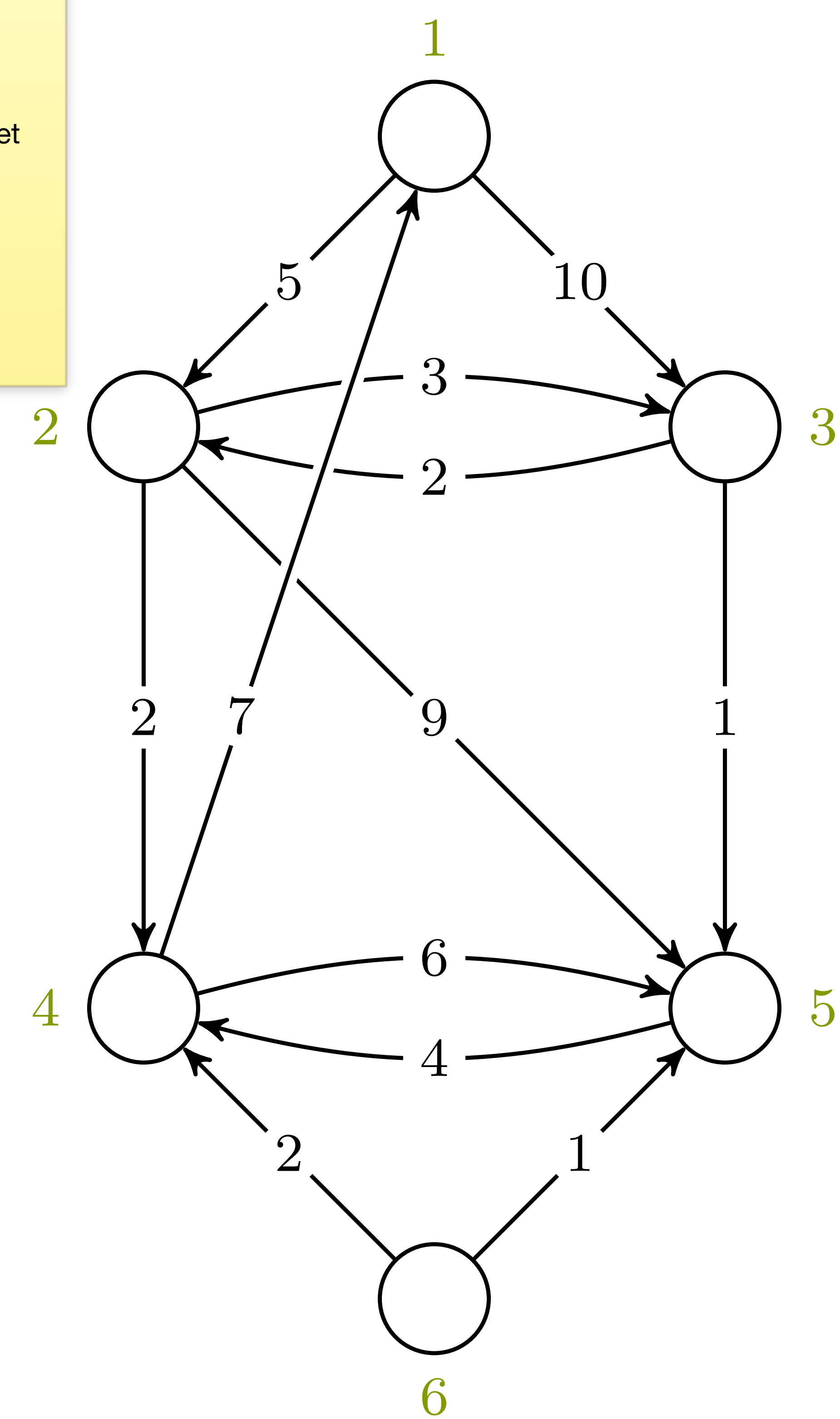
G	graf
w	vekting
s	startnode
S	ferdige
Q	pri-kø
u	fra-node
v	til-node

$u.d$ er rett; slakk alle ut-kanter

En liten forskjell fra boka: De farger en node svart idet den legges i S, mens jeg farger den svart idet den er ferdigbehandlet – dvs., etter vi har slakket ut-kantene.

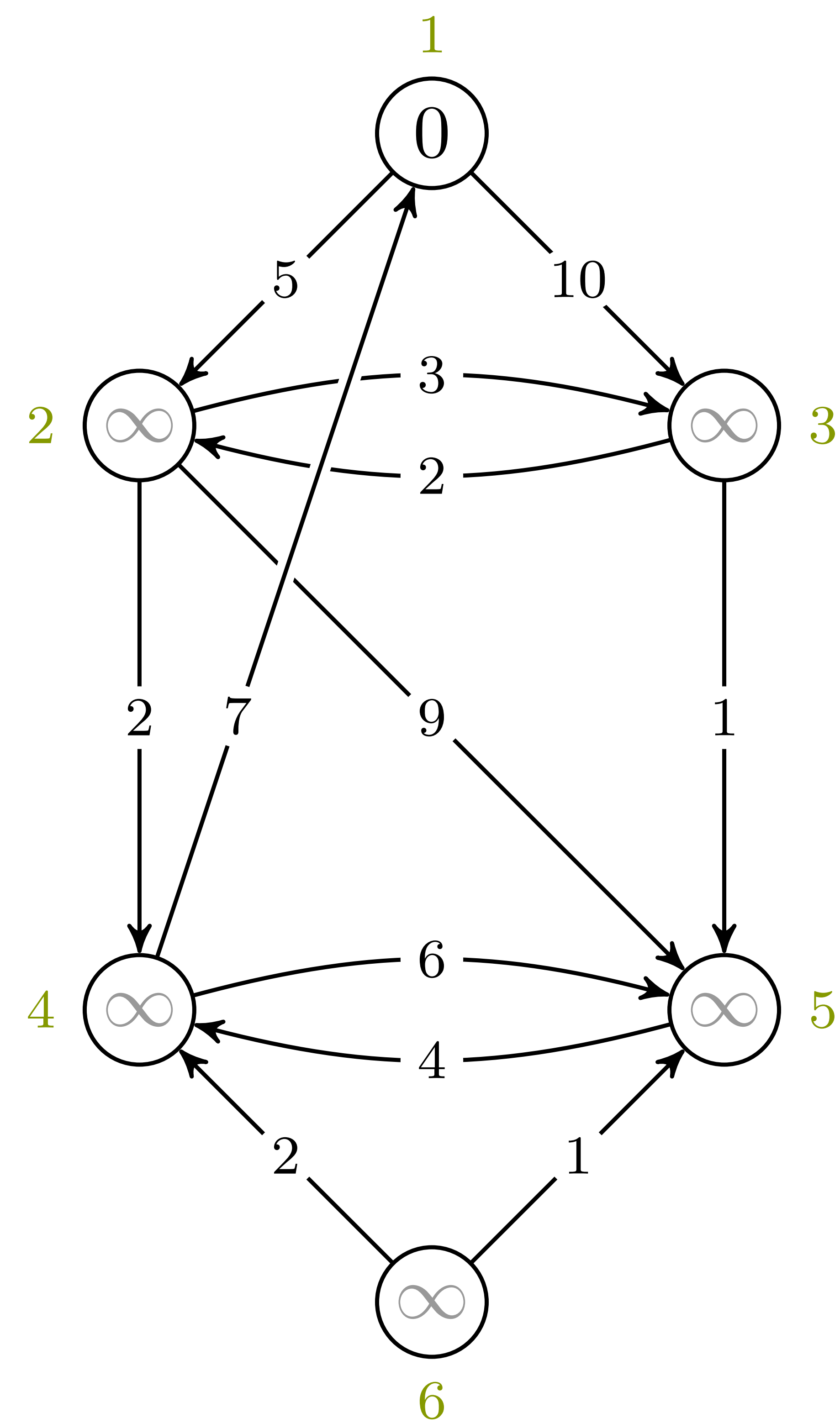
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$u, v = -, -$



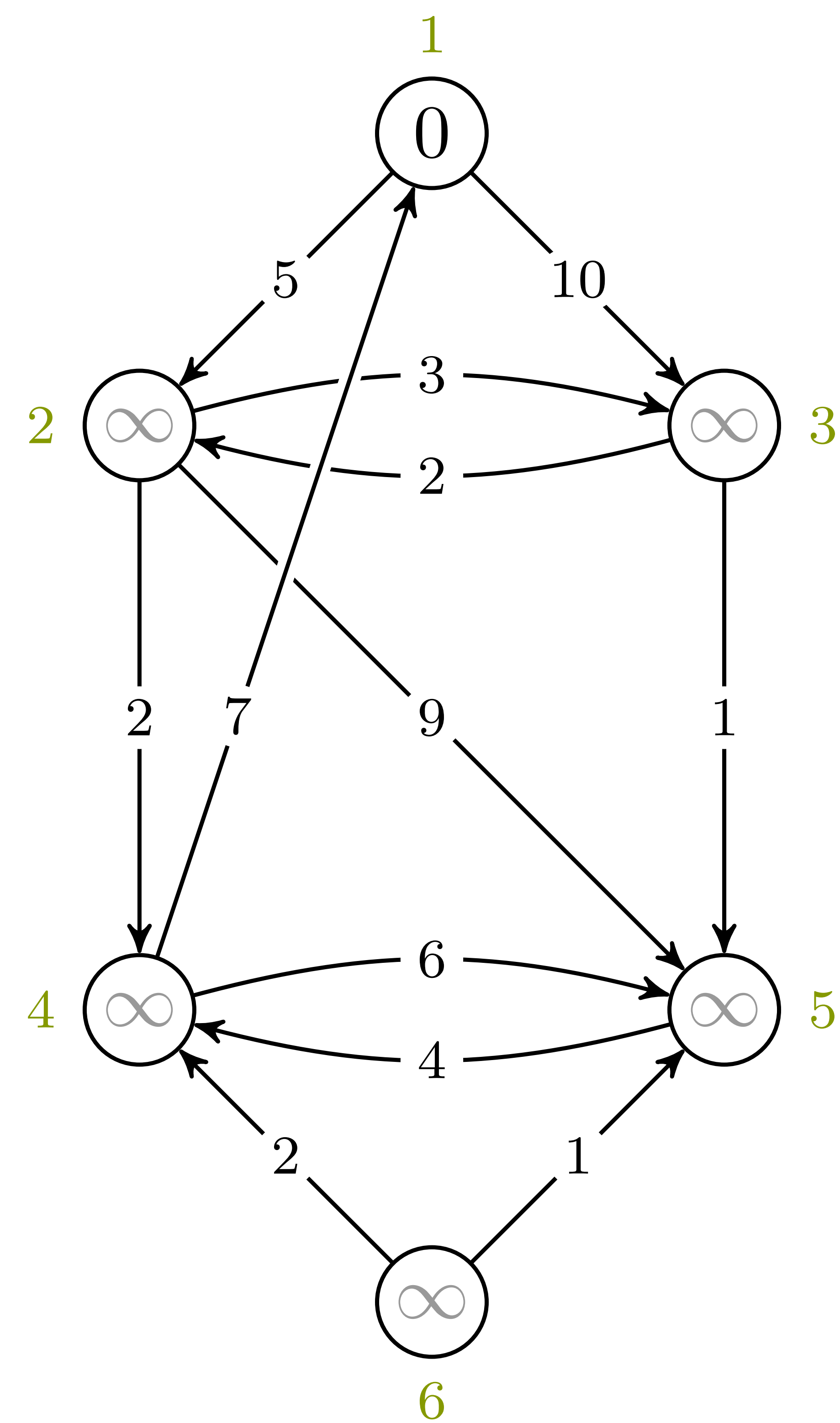
```
DIJKSTRA(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u, v, w)
```

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



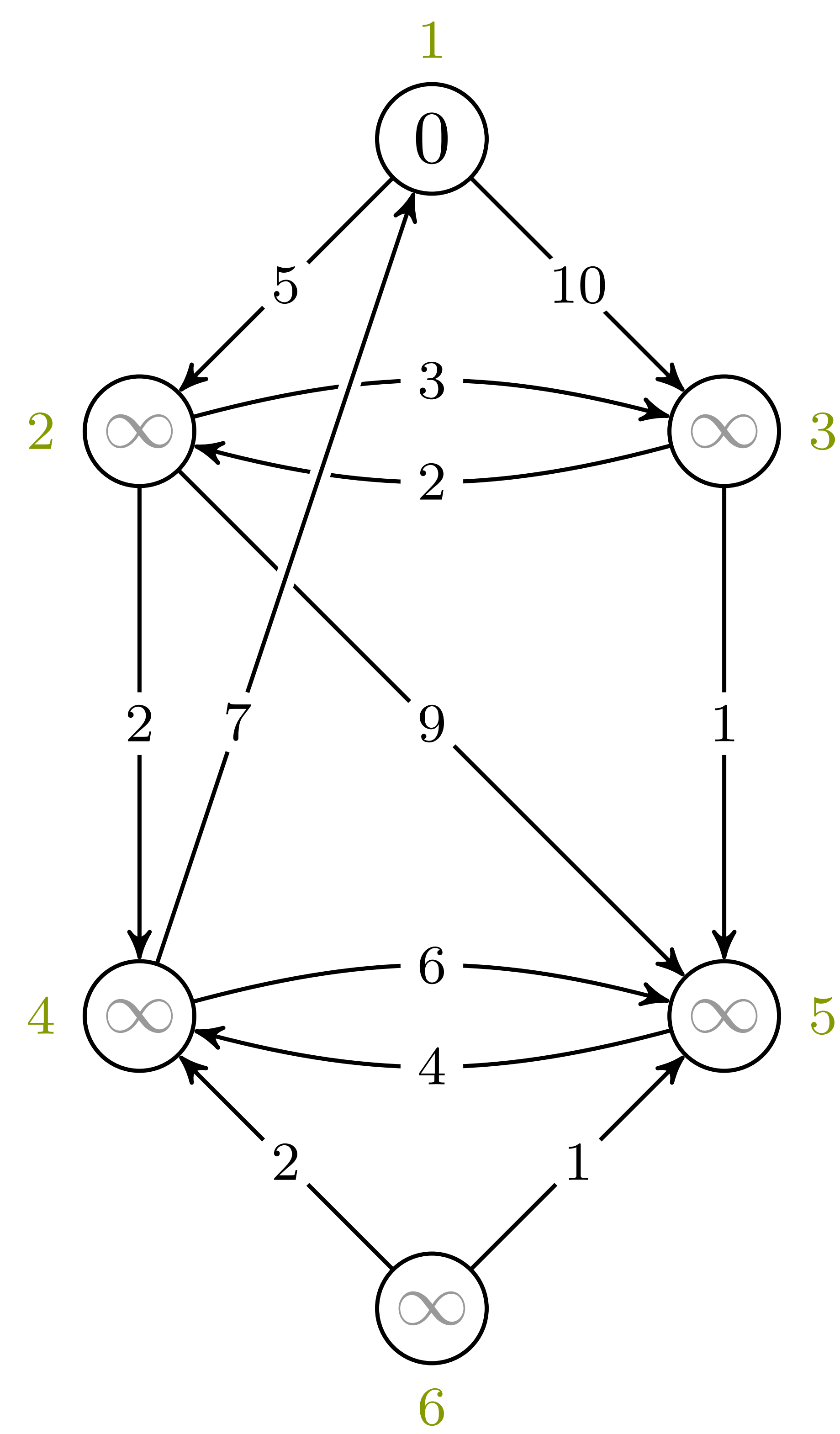
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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

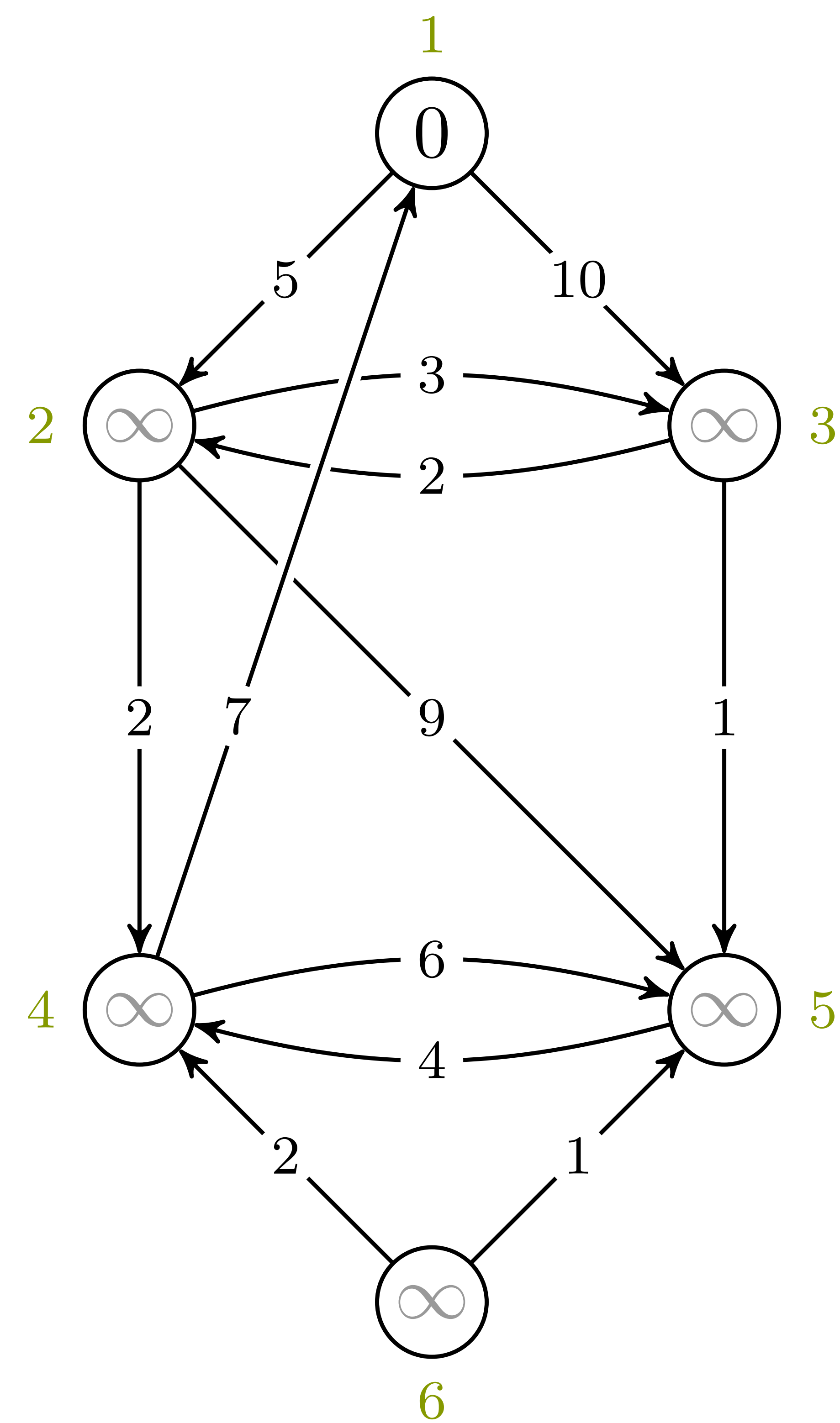


```
Dijkstra(G, w, s)
Initialize-Single-Source(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```

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

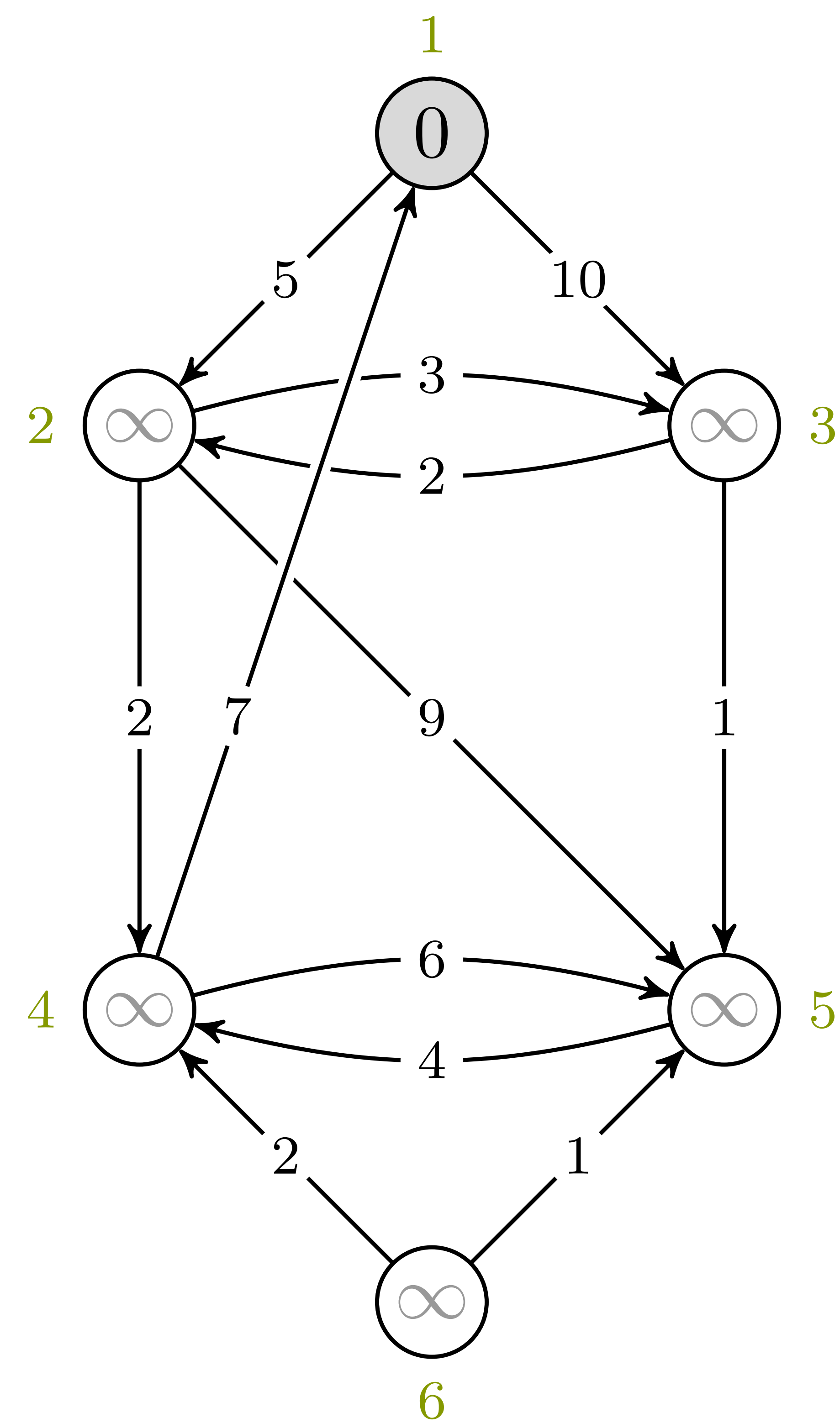


```
DIJKSTRA(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u, v, w)
```



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

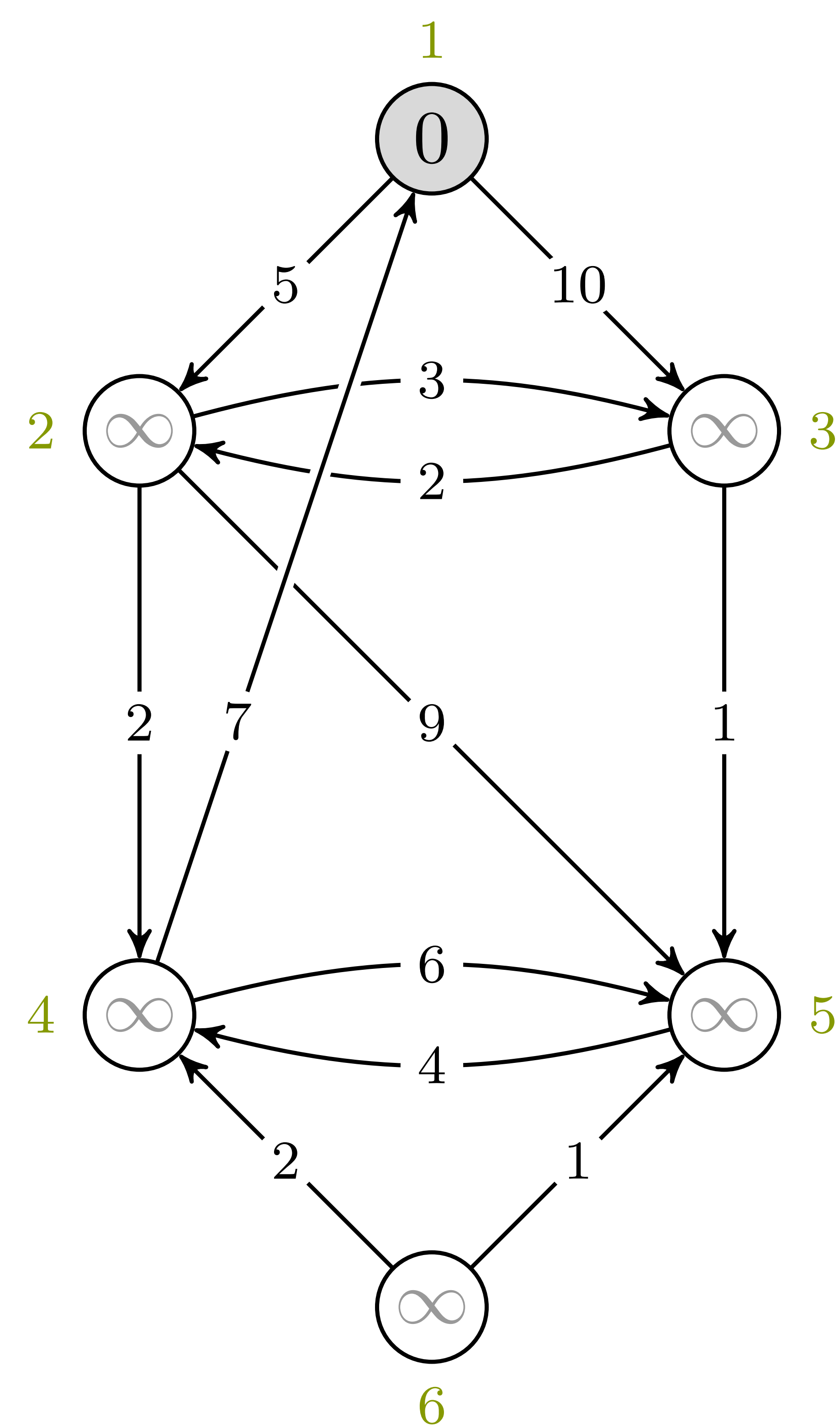
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



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

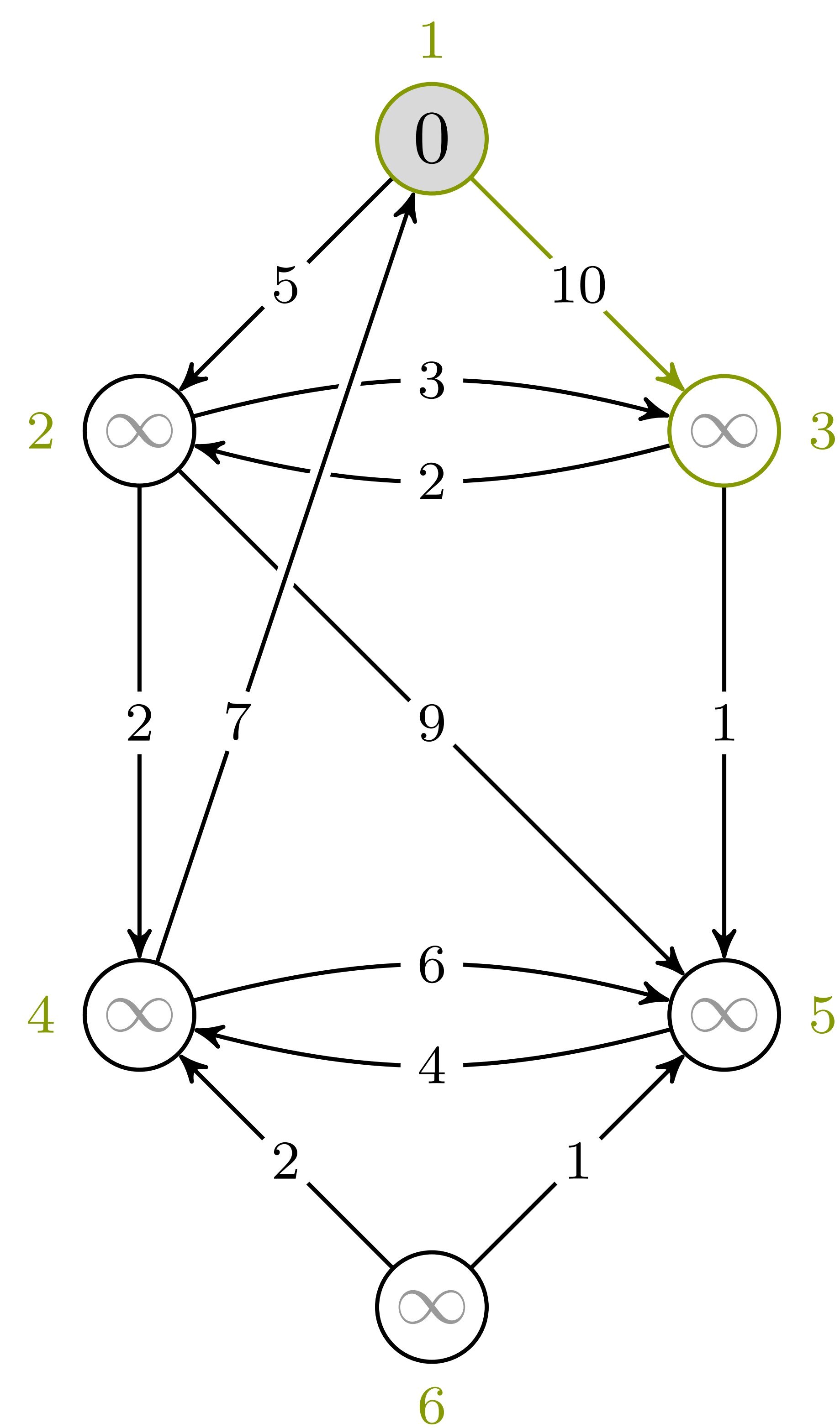
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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



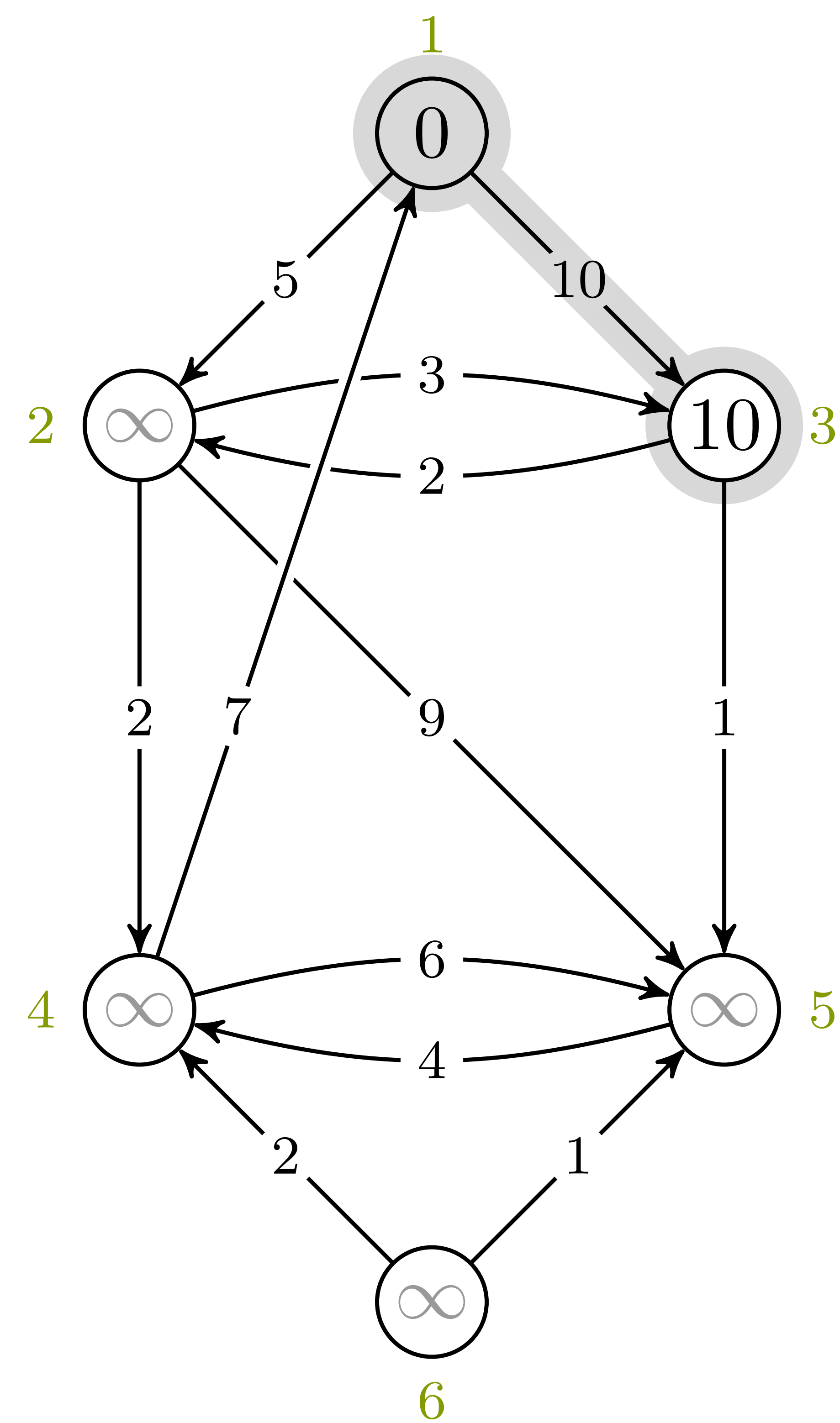
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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



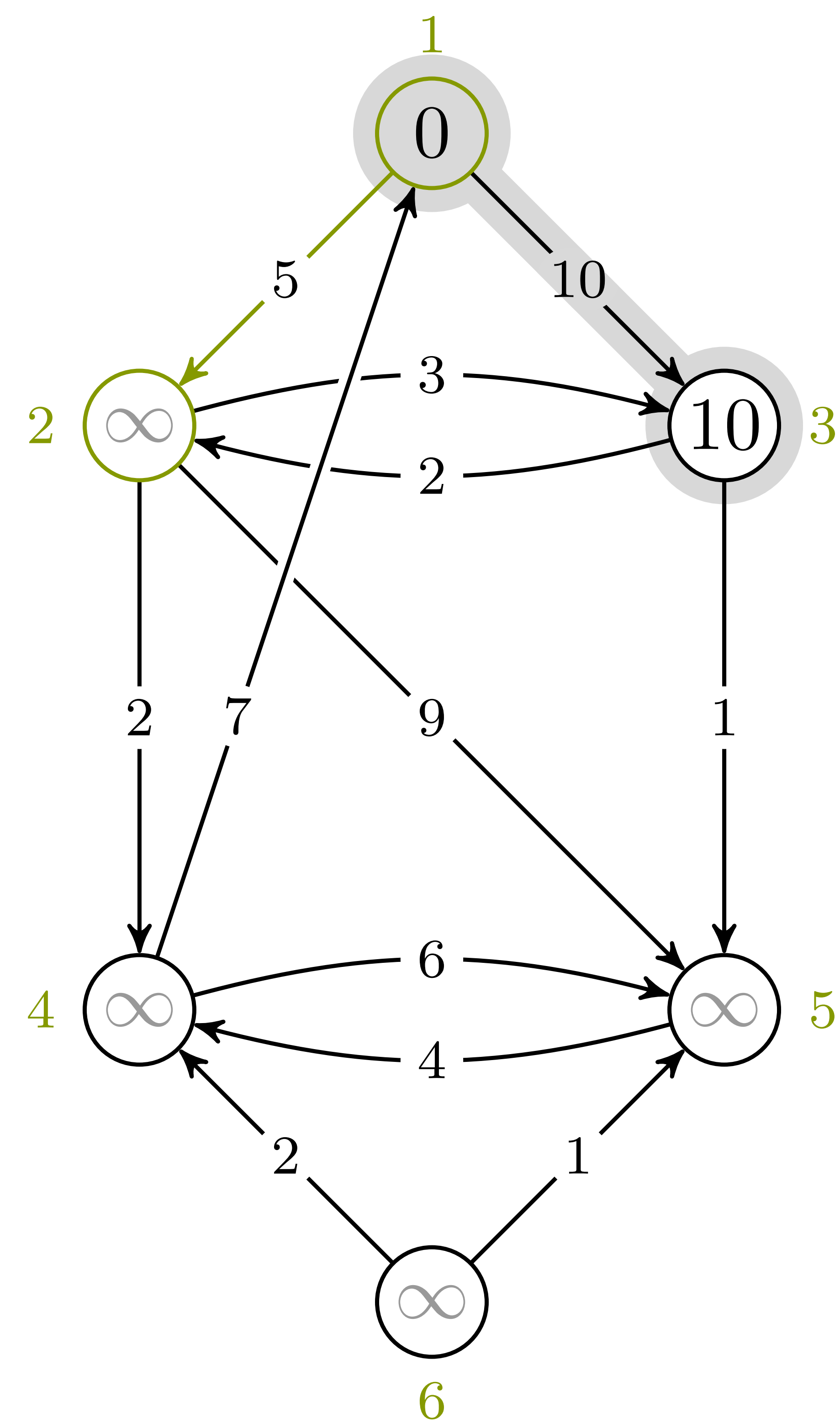
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

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



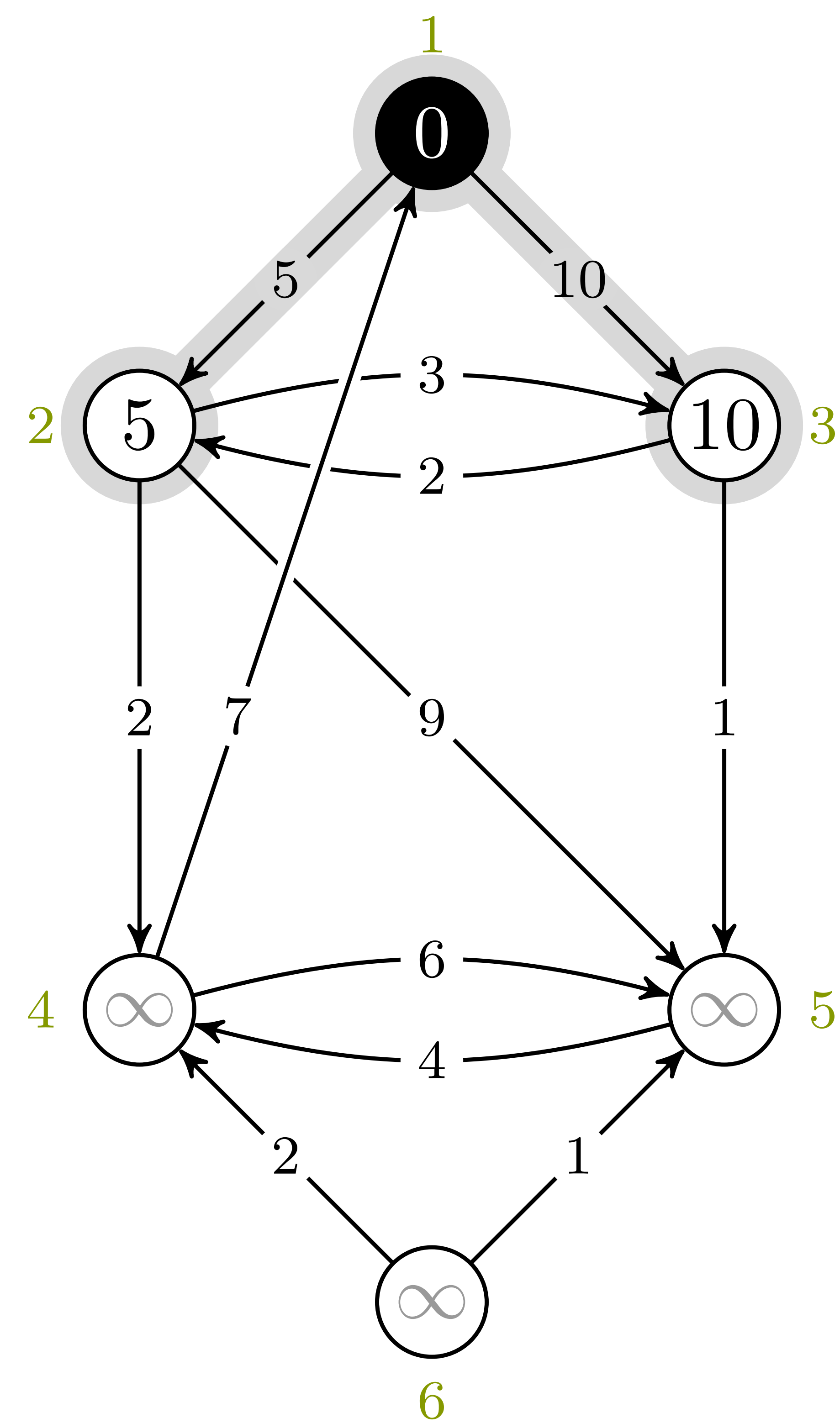
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v = 1, 2$



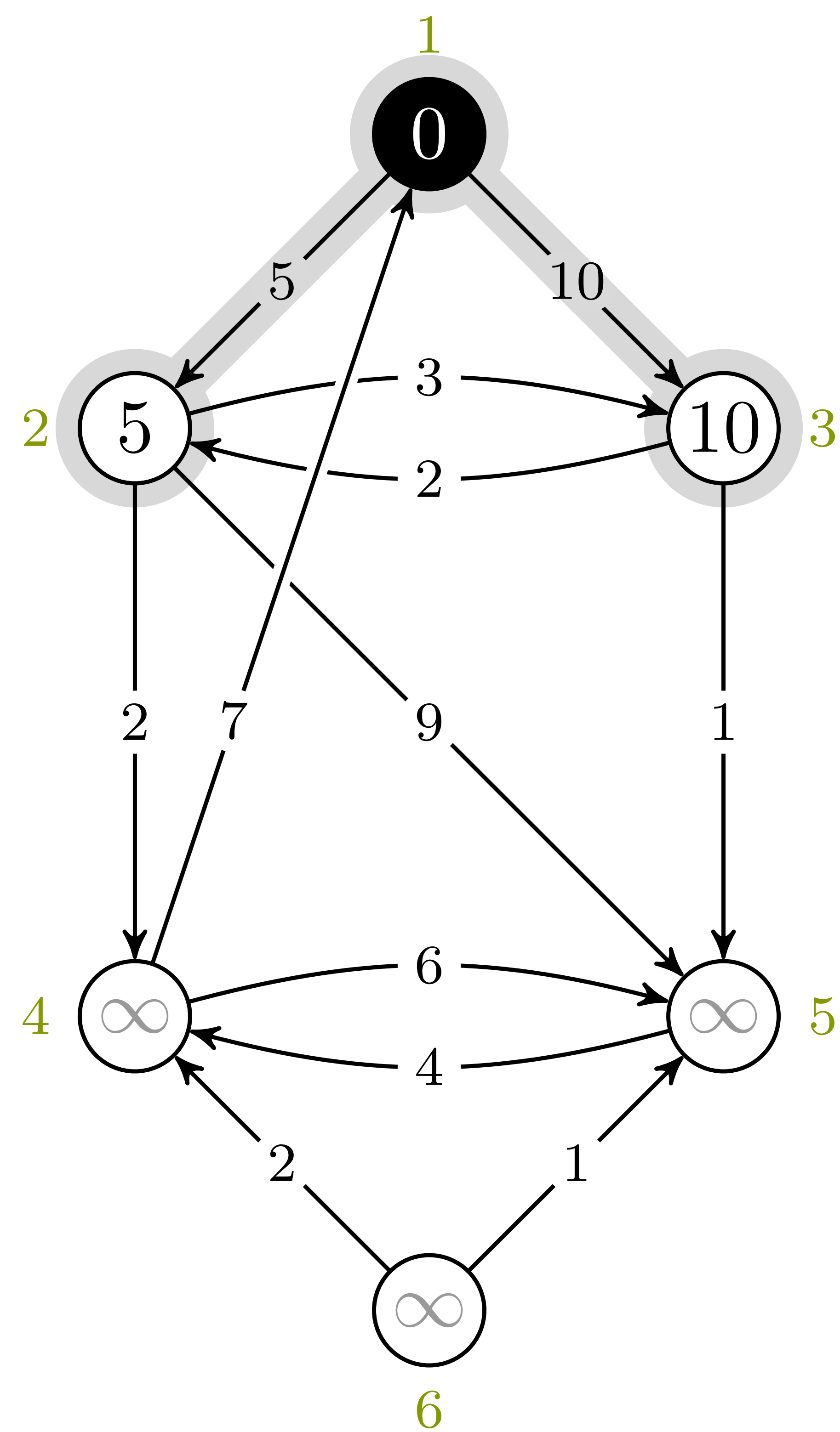
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v = -, -$



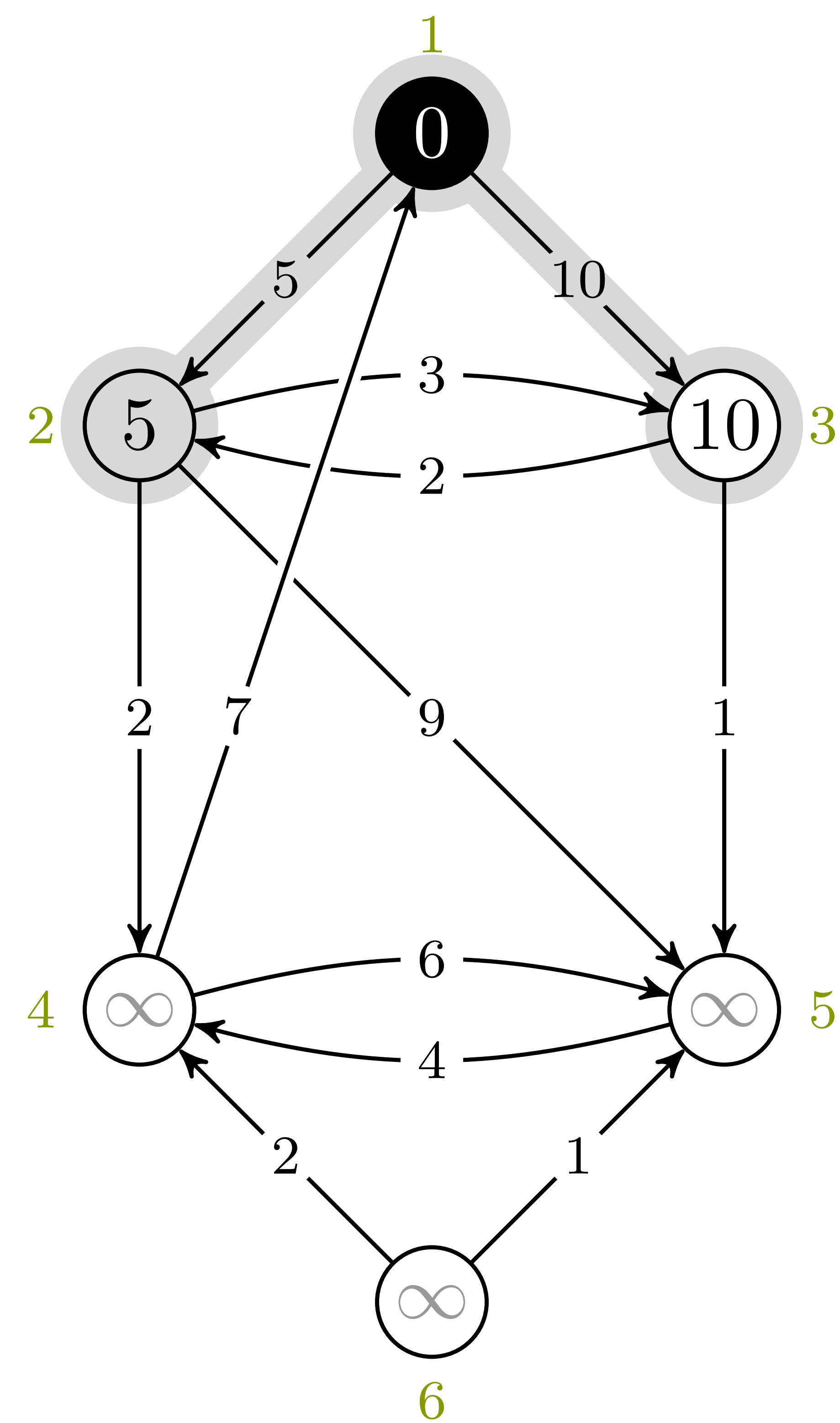
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v \ = \ -, -$

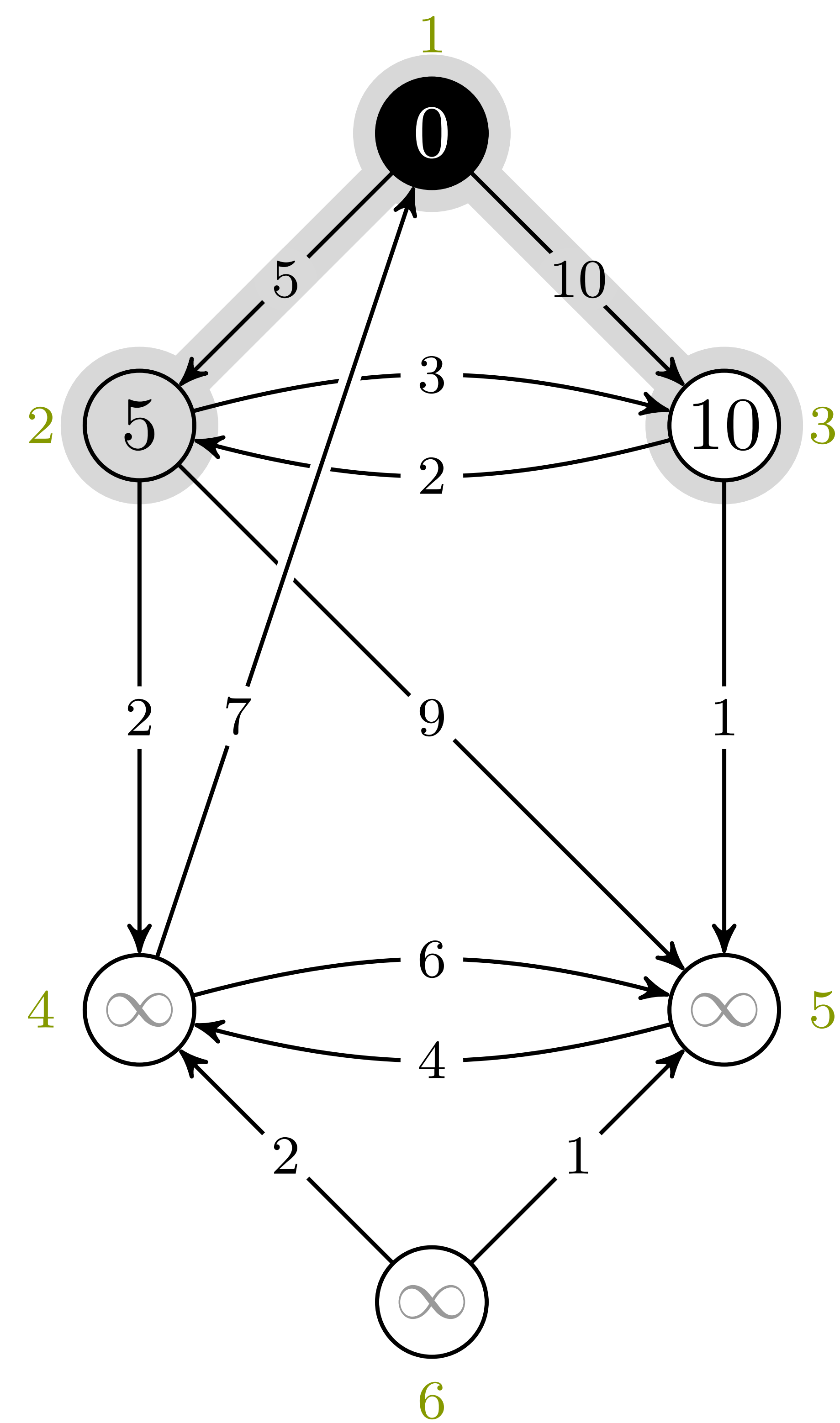


```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
                RELAX(u, v, w)
```

$u, v \;=\; 2, \text{-}$

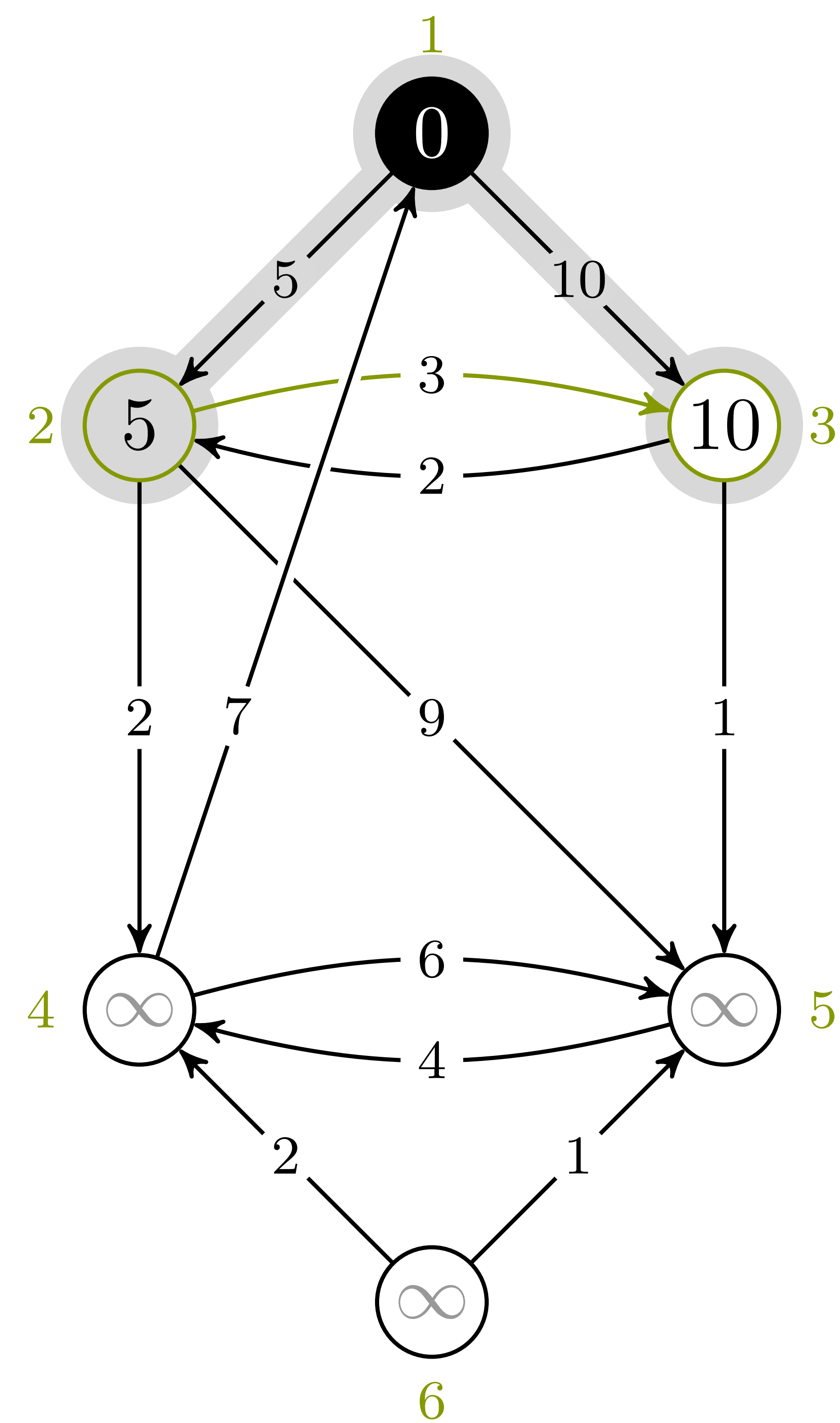


```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



$u, v \ = \ 2, \text{-}$

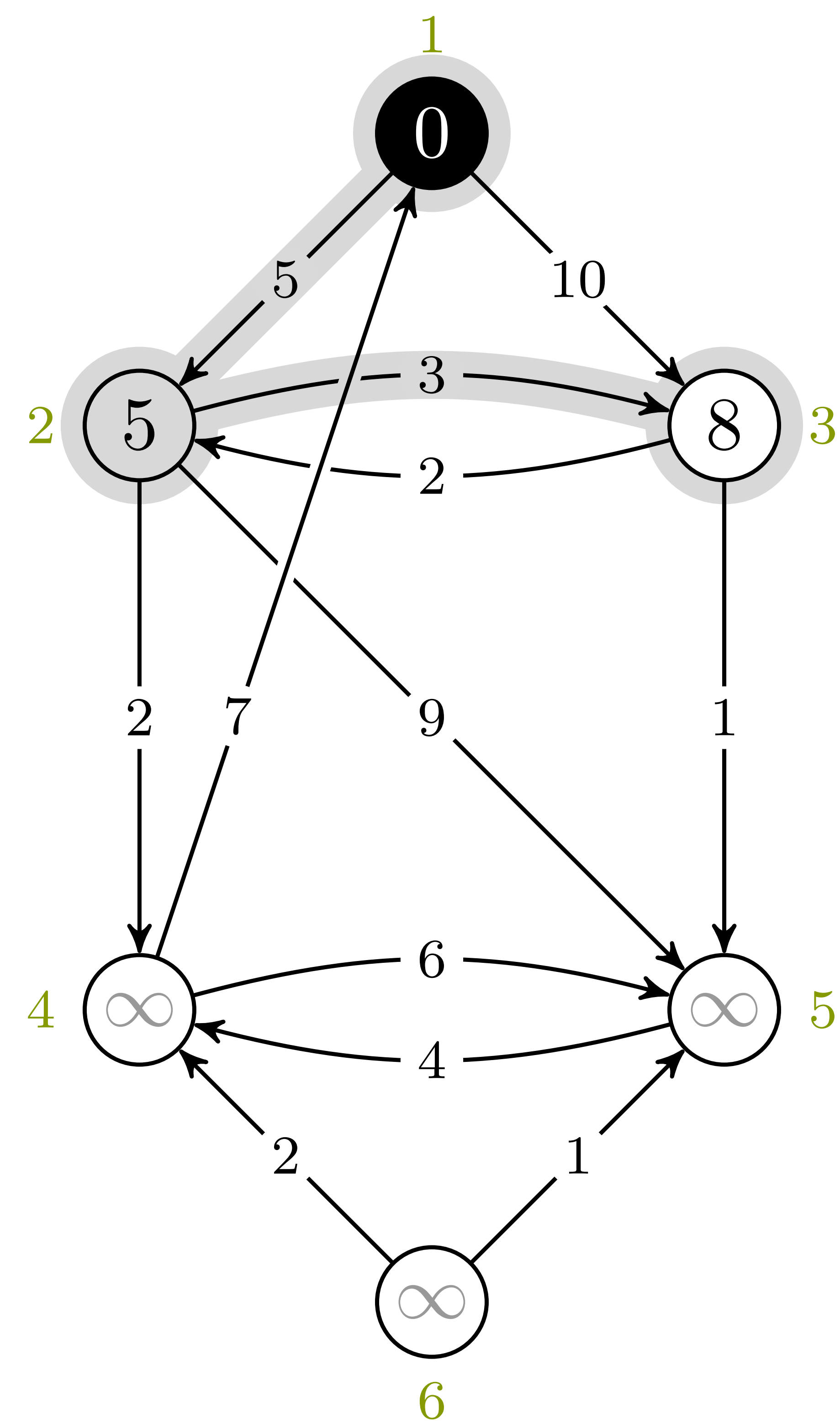
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



$u, v \;=\; 2, 3$

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v \ = \ 2, 3$



DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)

$S = \emptyset$

$Q = G.V$

while $Q \neq \emptyset$

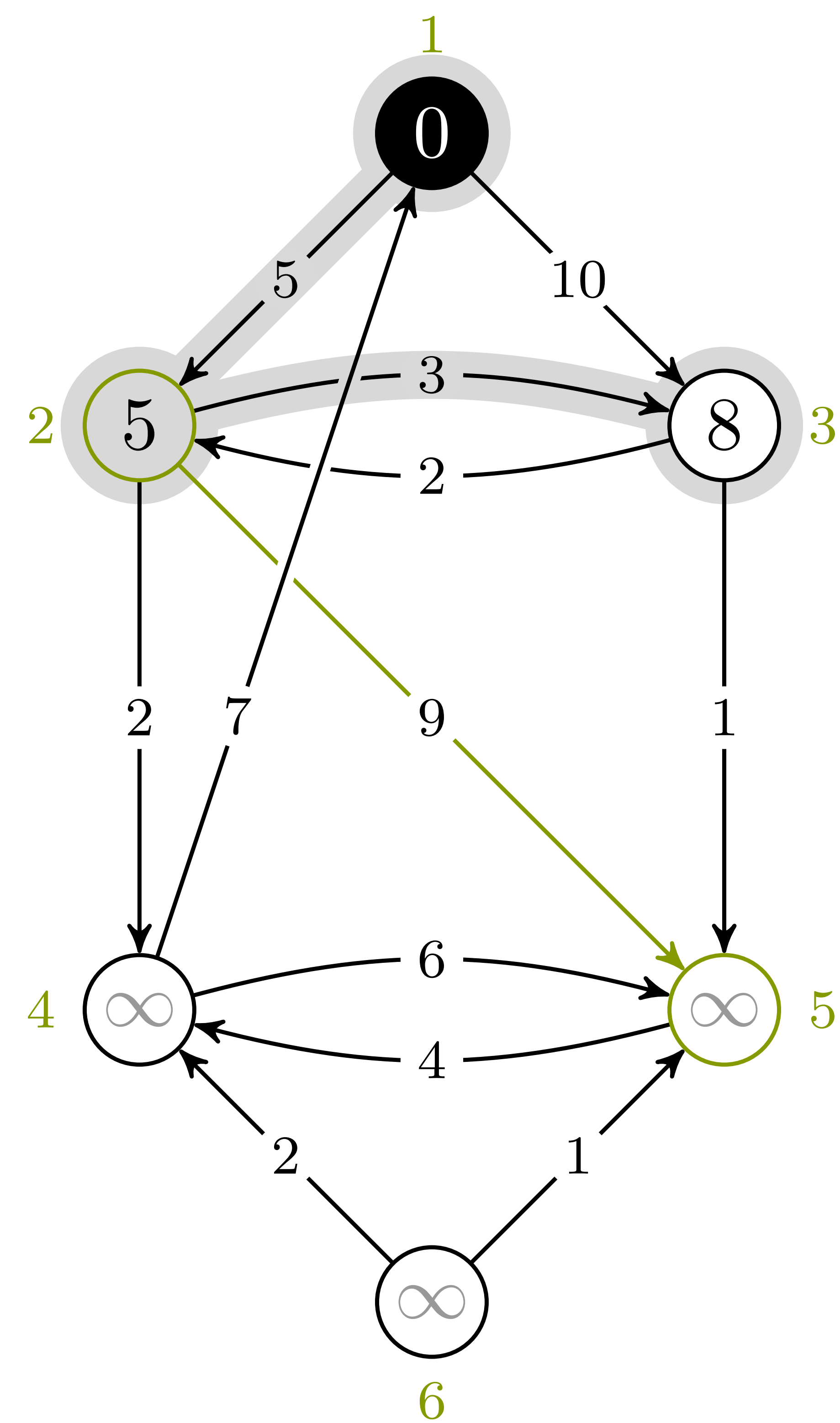
  $u =$ EXTRACT-MIN(Q)

  $S = S \cup \{u\}$

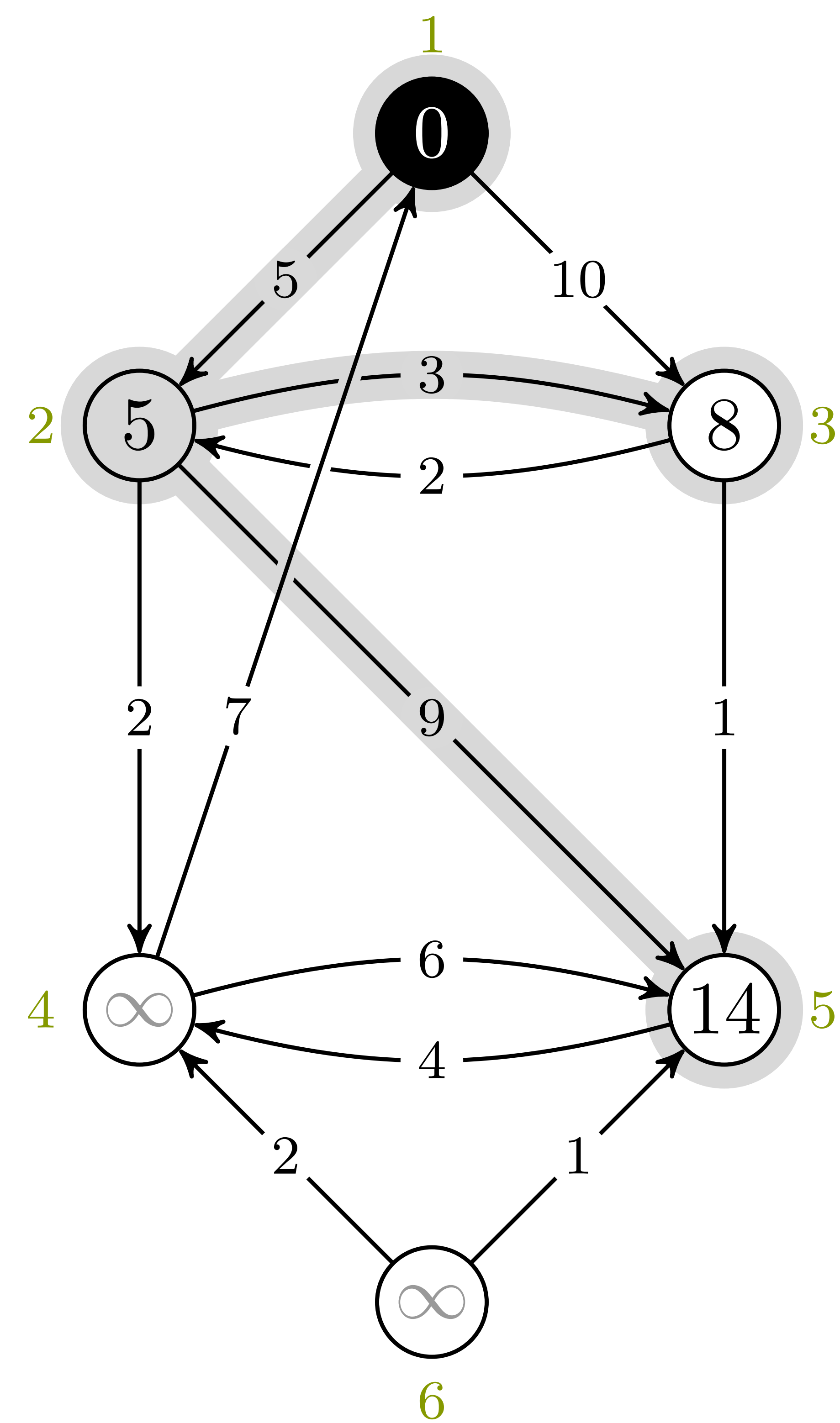
  for each vertex $v \in G.Adj[u]$

    RELAX(u, v, w)

$u, v \; = \; 2, 5$



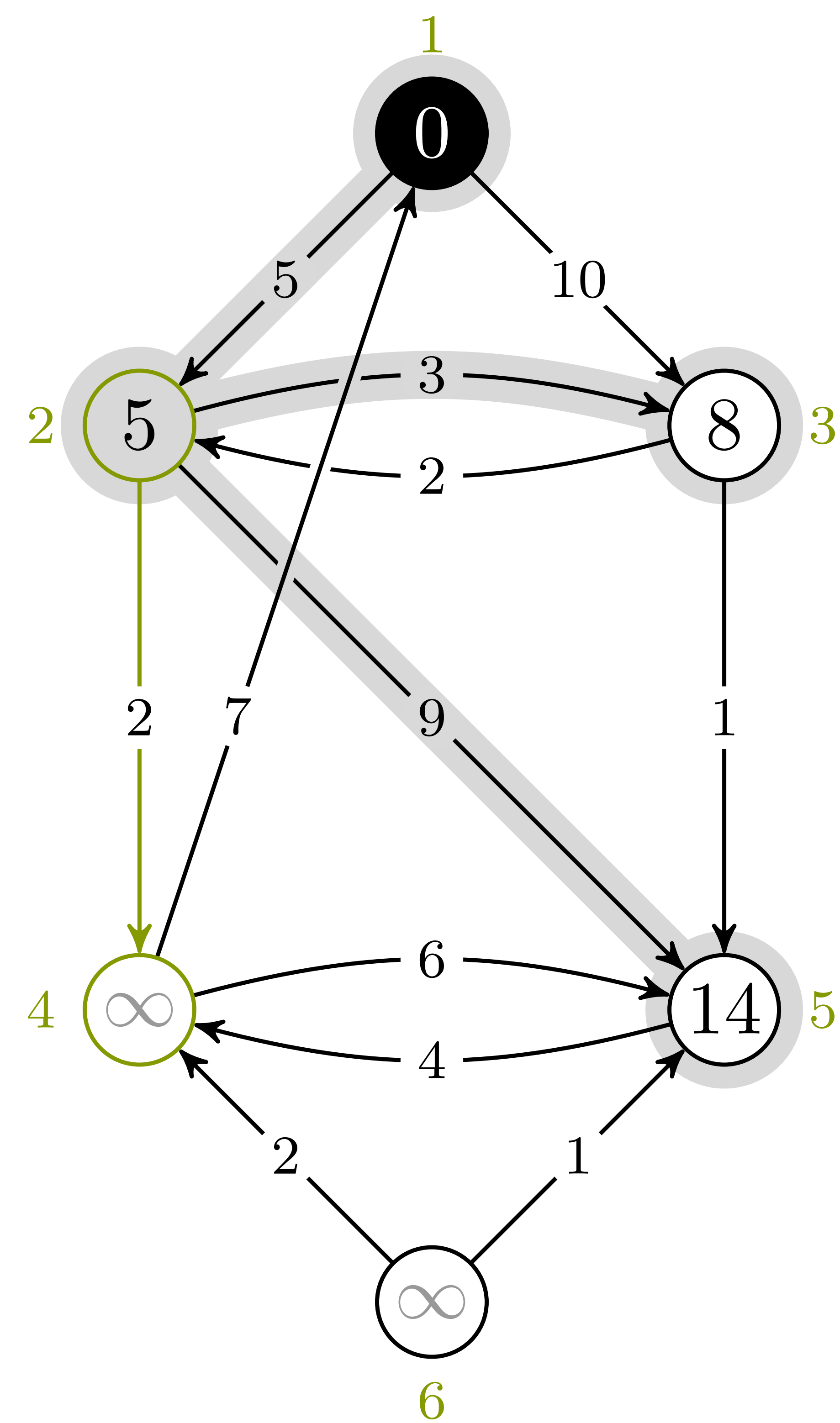
```
Dijkstra(G, w, s)
Initialize-Single-Source(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```



$u, v \ = \ 2, 5$

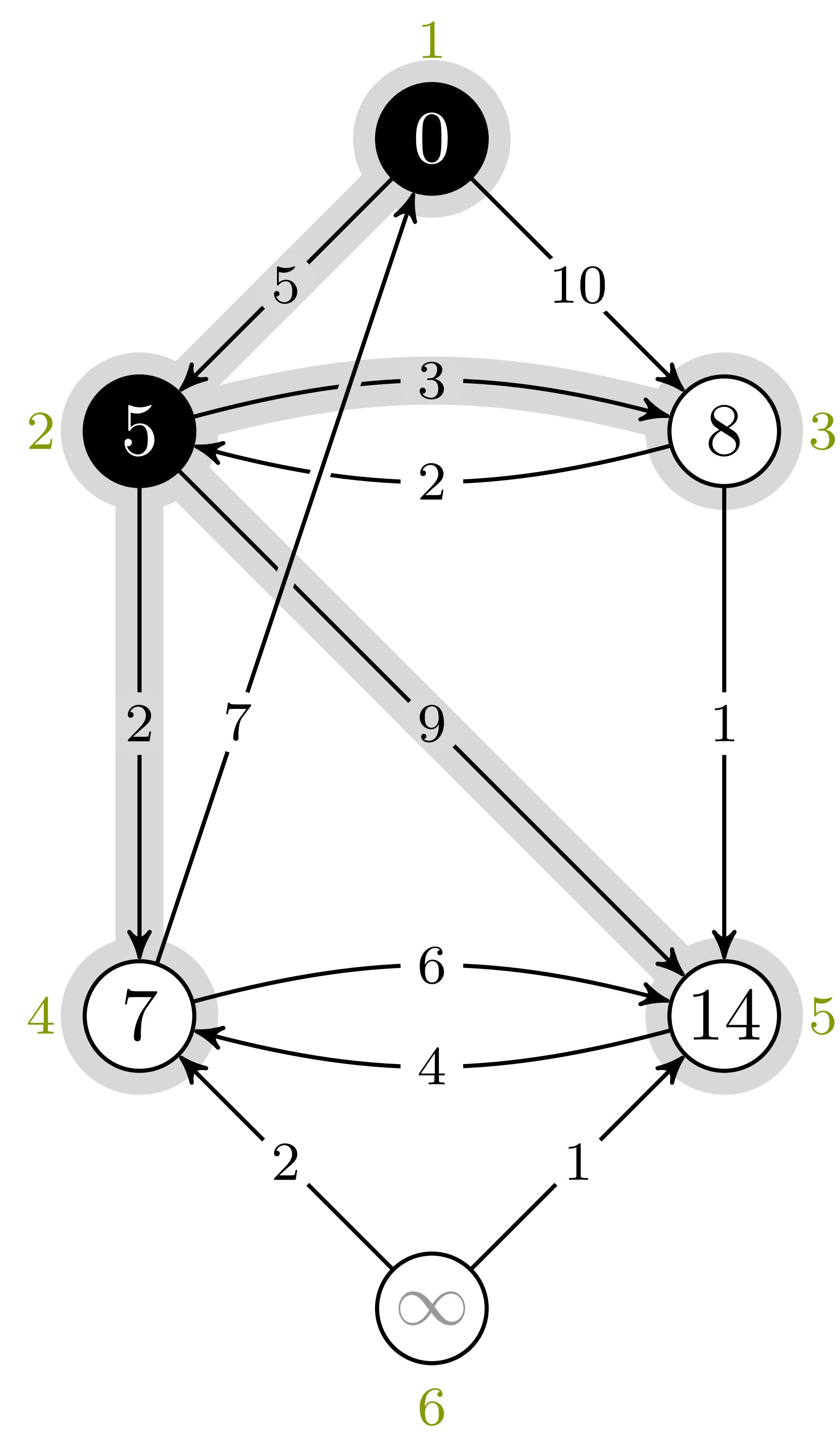
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v \;=\; 2, 4$

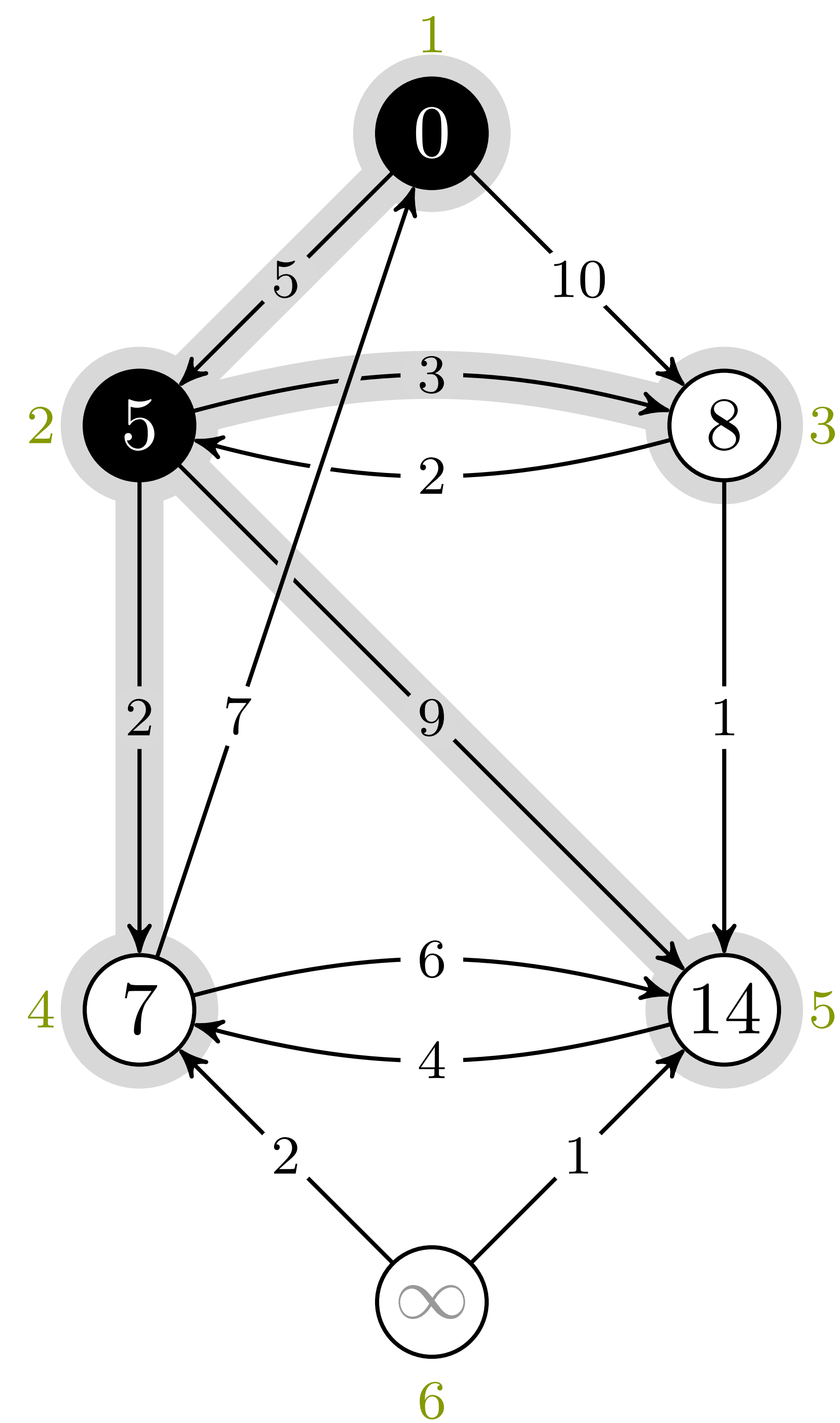


```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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



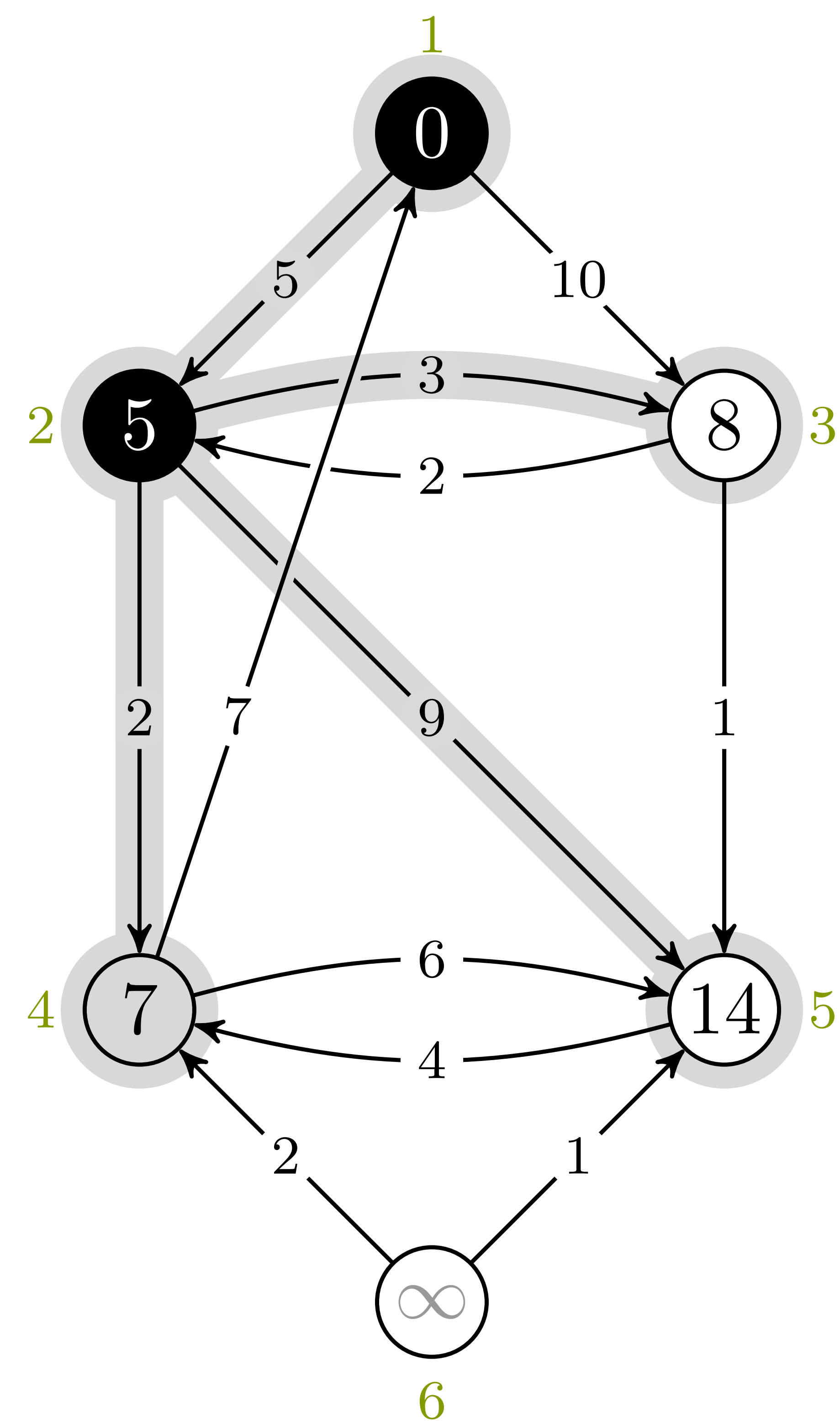
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



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

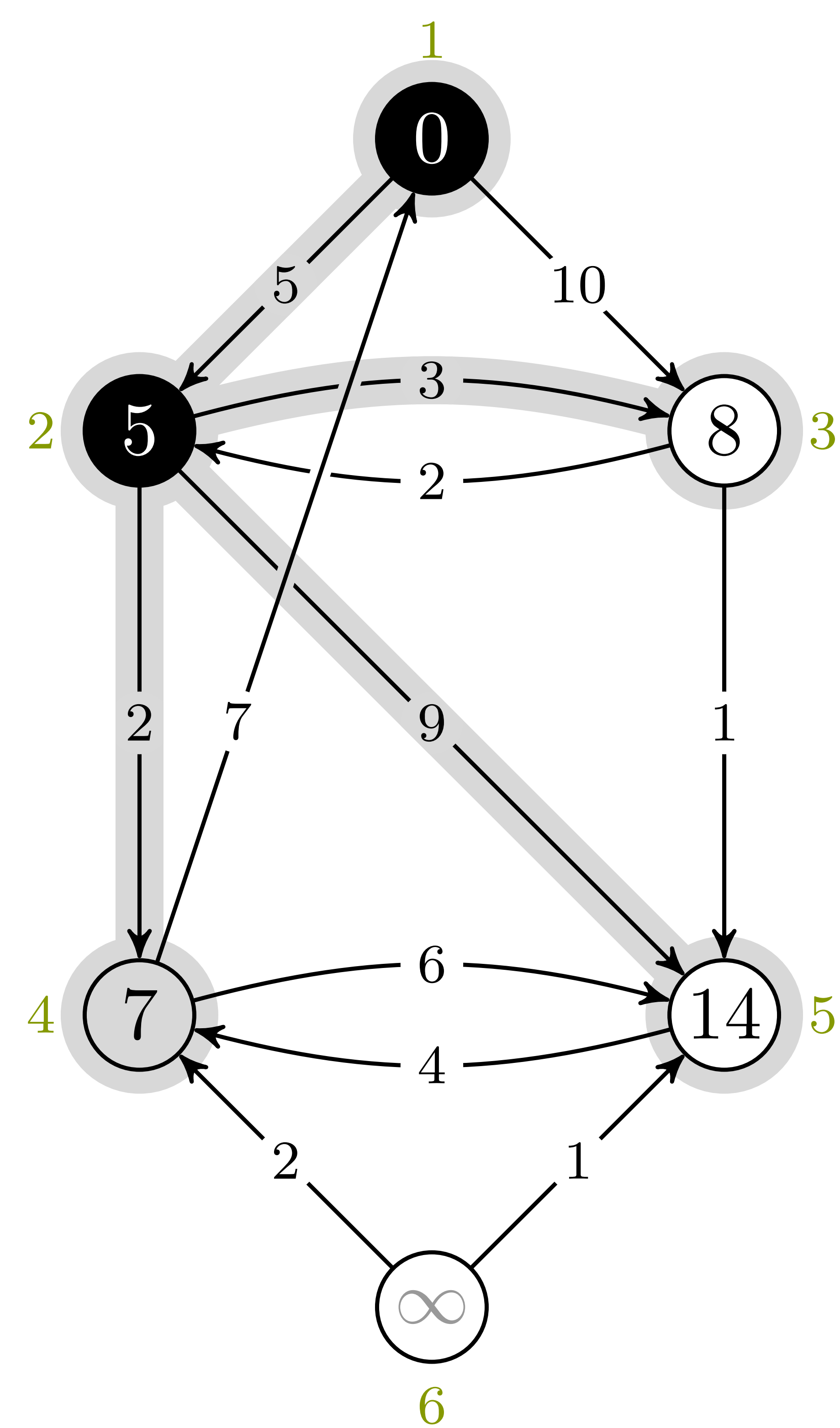
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v \ = \ 4, \text{-}$



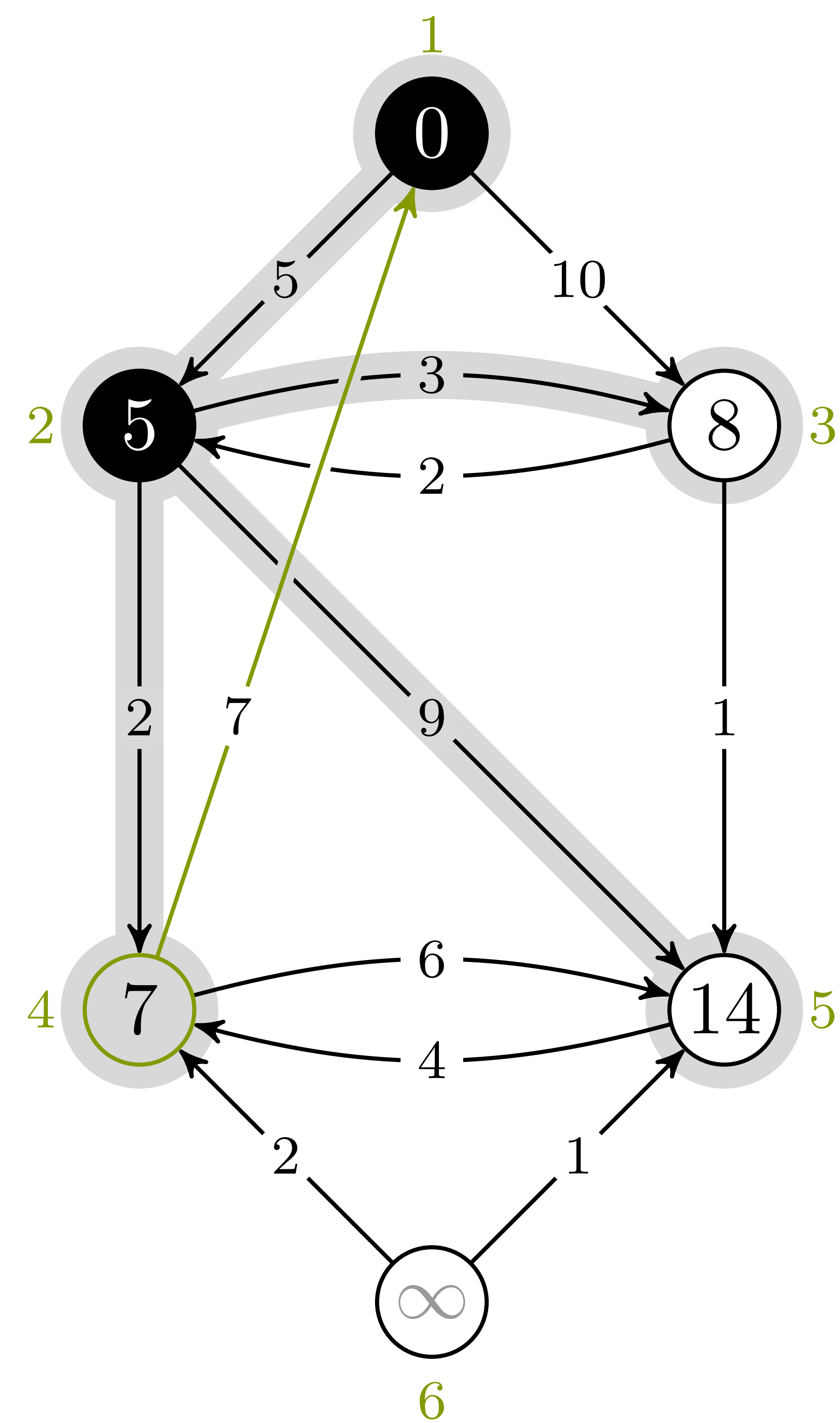
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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



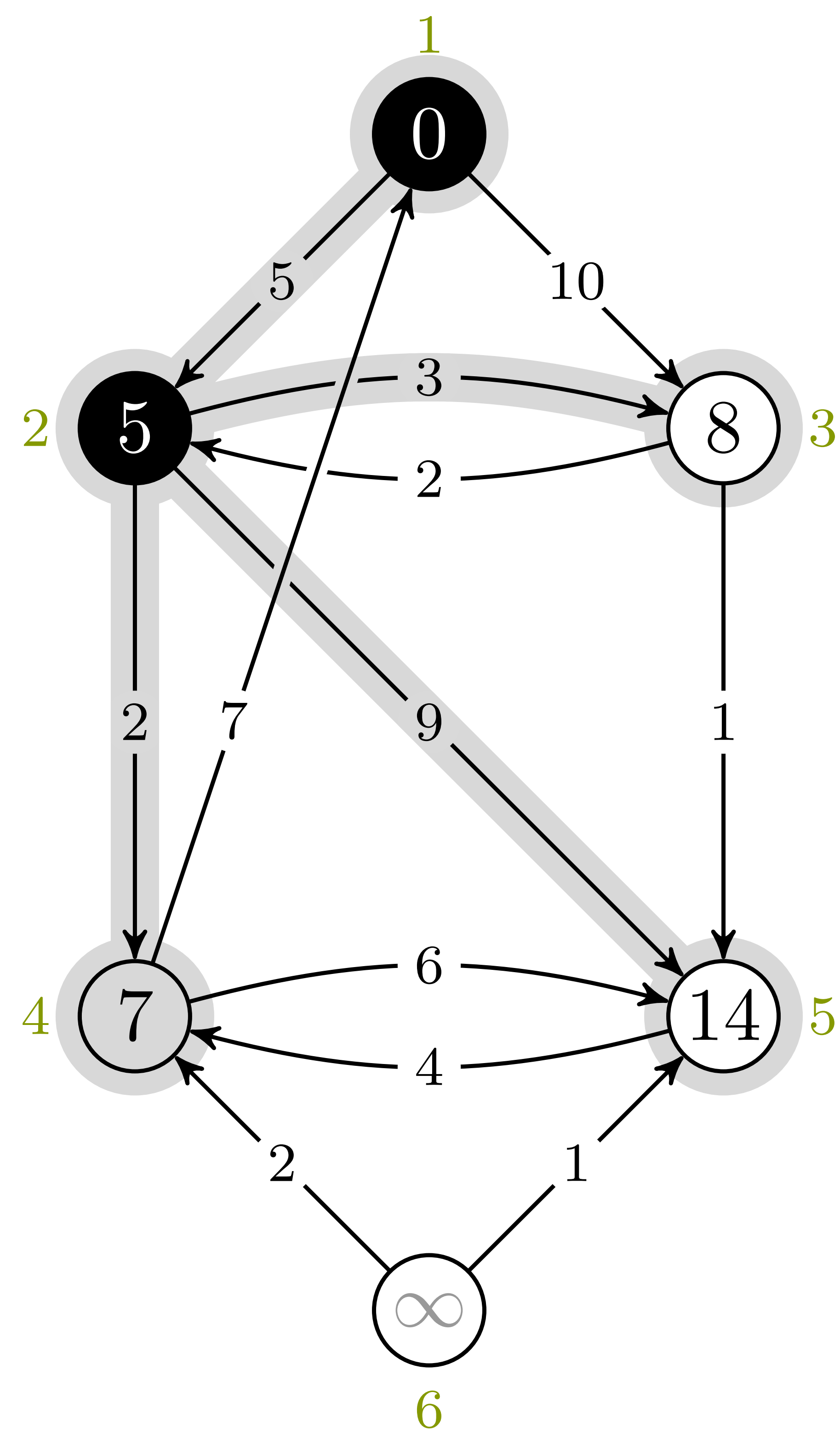
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v \;=\; 4, 1$

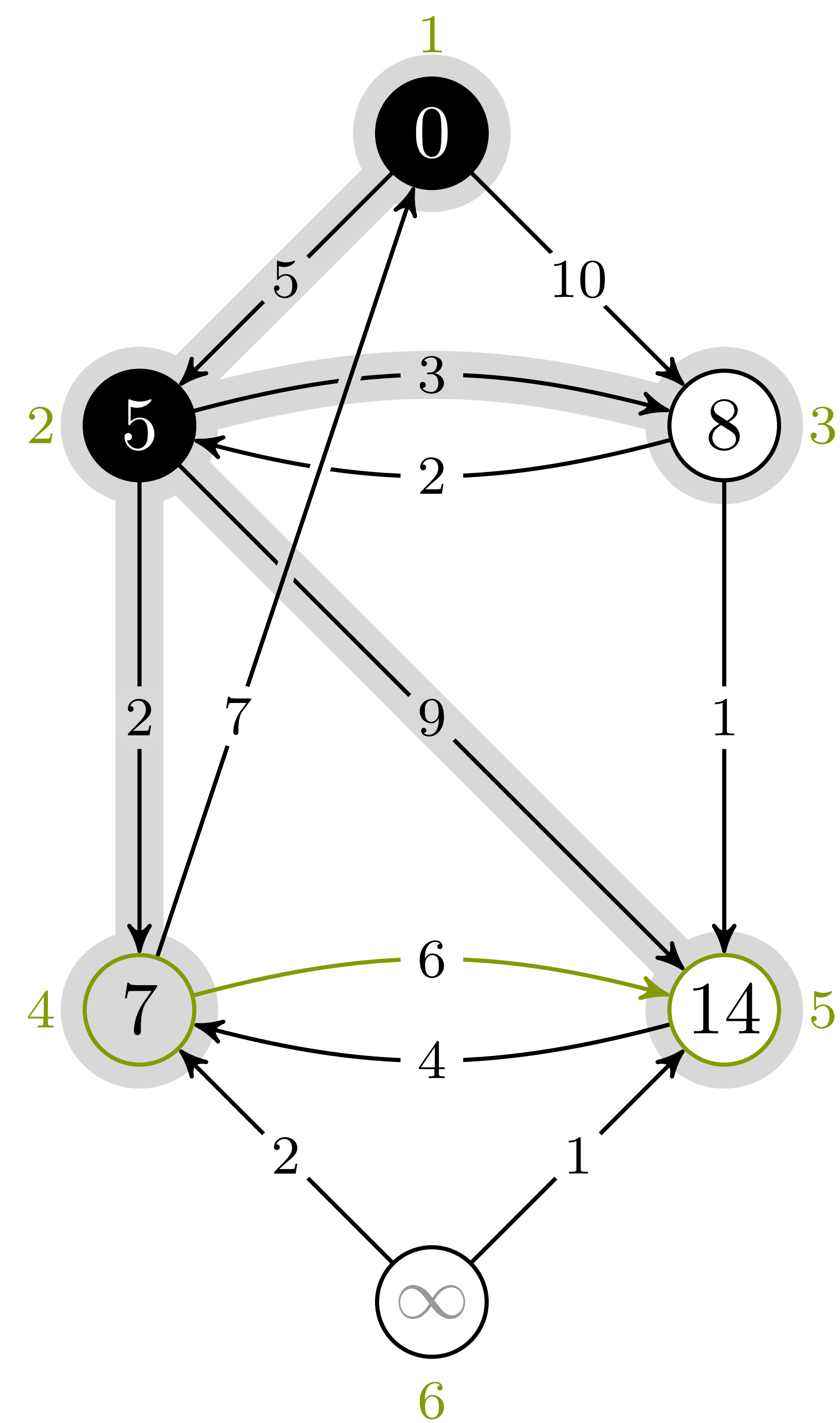


```
Dijkstra(G, w, s)
  Initialize-Single-Source(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
        u = Extract-Min(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
              Relax(u, v, w)
```

$u, v \;=\; 4, 1$



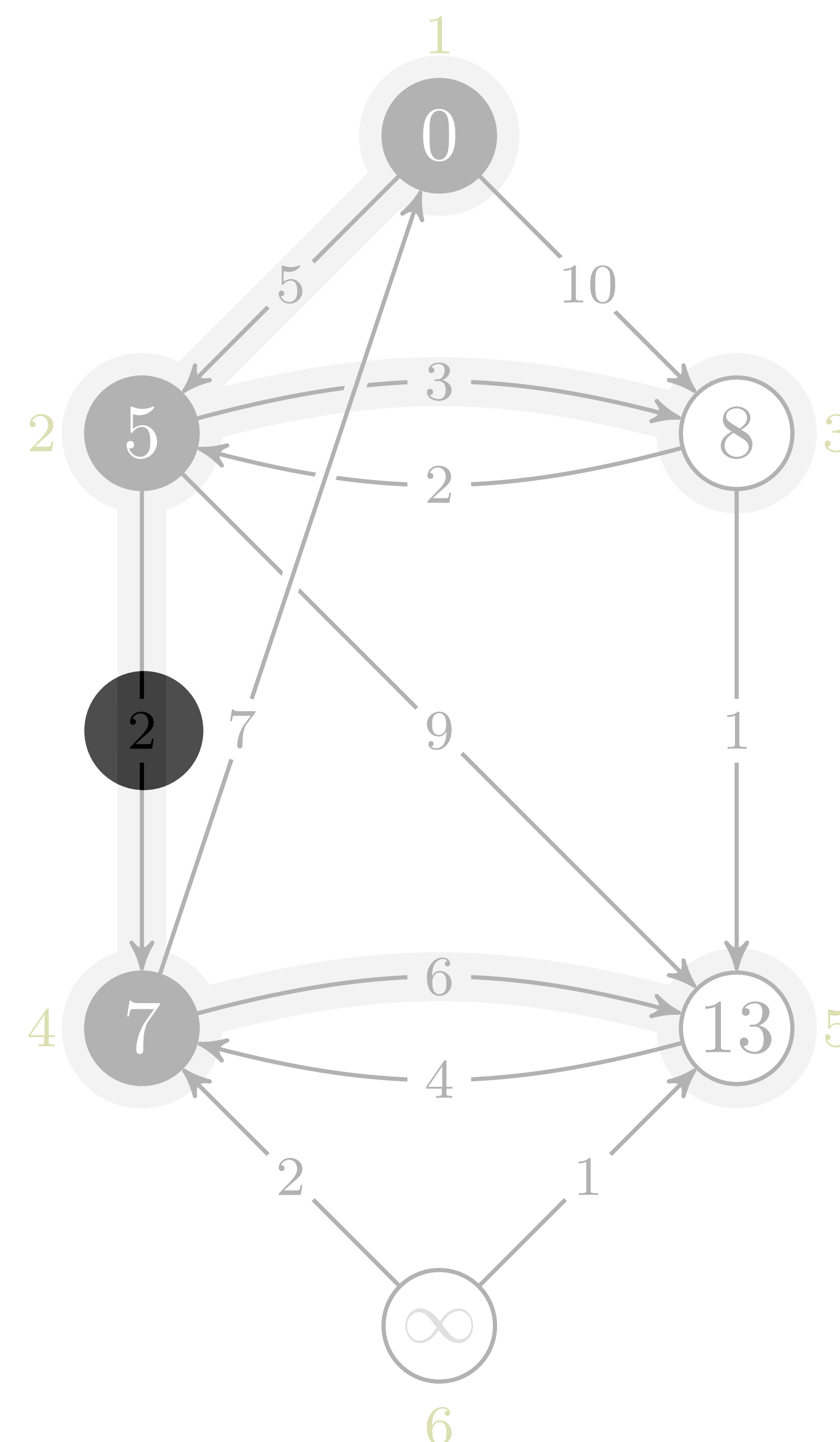
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



$u, v \;=\; 4, 5$

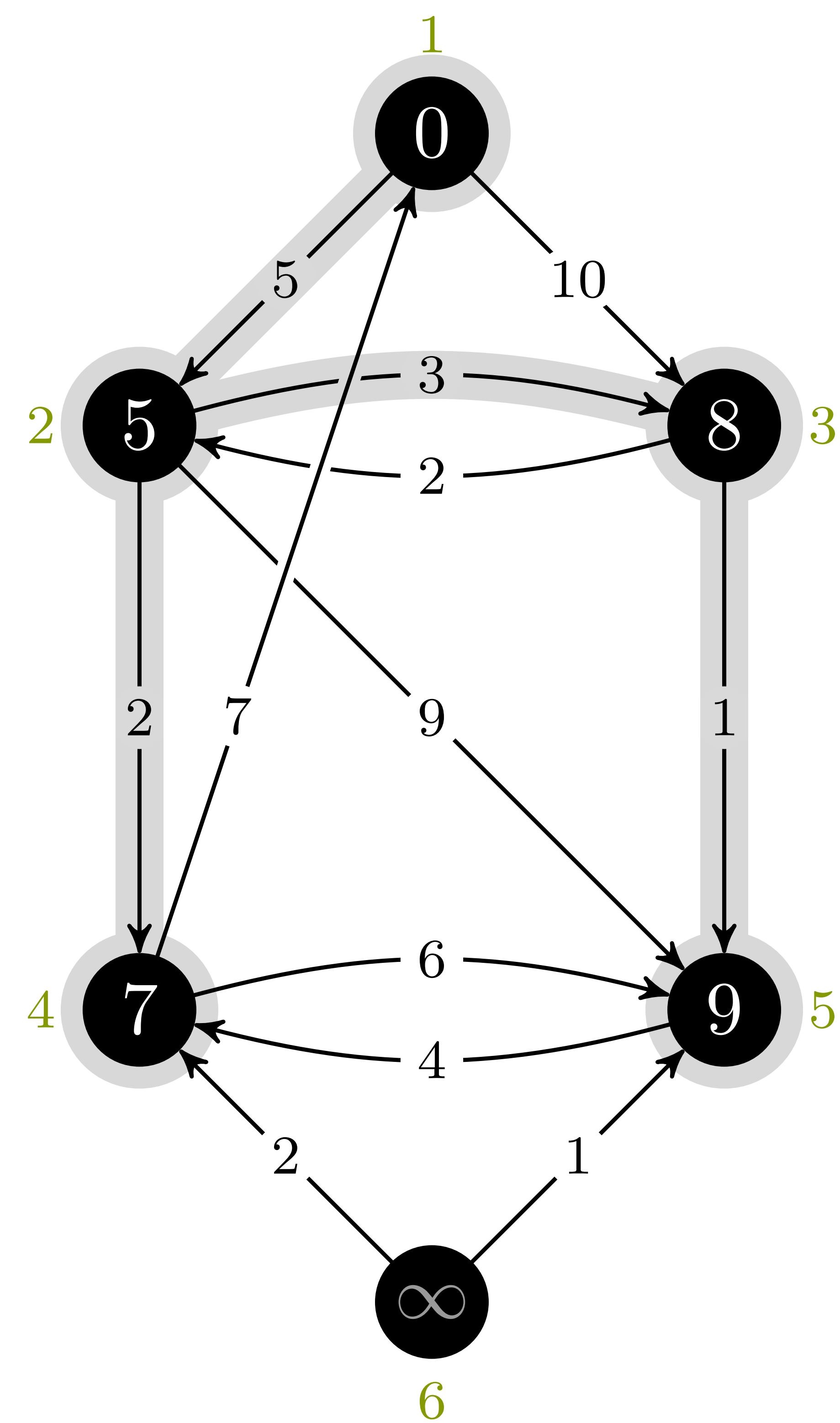
DIJKSTRA(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



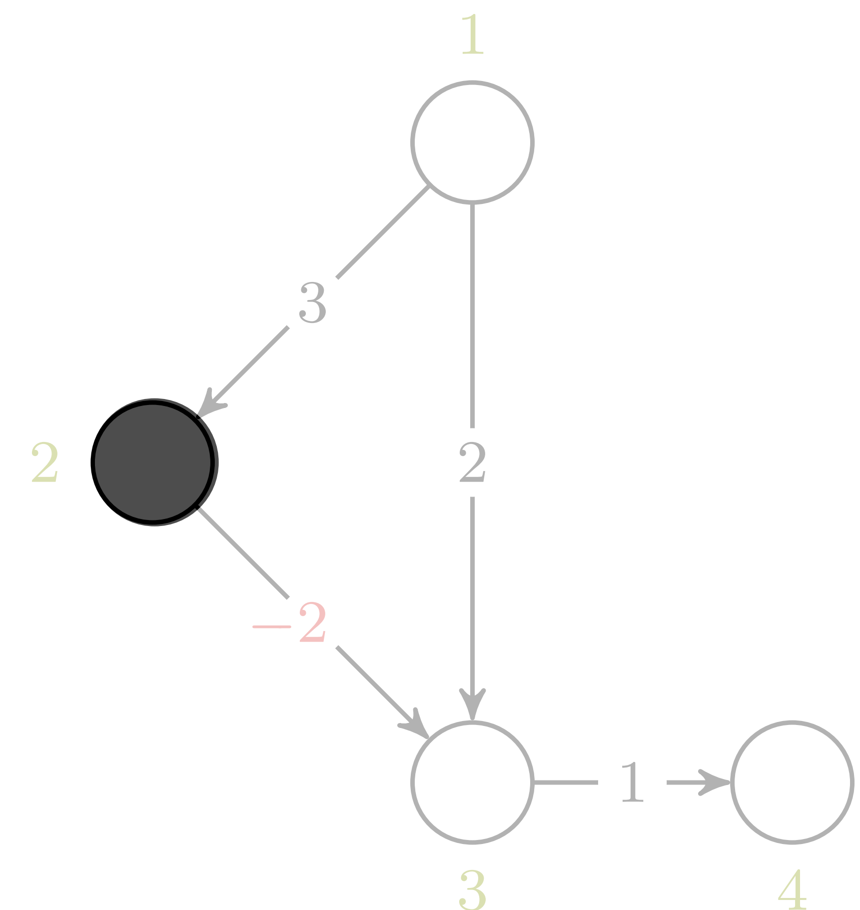
$u, v \;=\;$ -, -

Dijkstra(G, w, s)
  Initialize-Single-Source(G, s)
  $S = \emptyset$
  $Q = G.V$
  while $Q \neq \emptyset$
    $u =$ Extract-Min(Q)
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
      Relax(u, v, w)



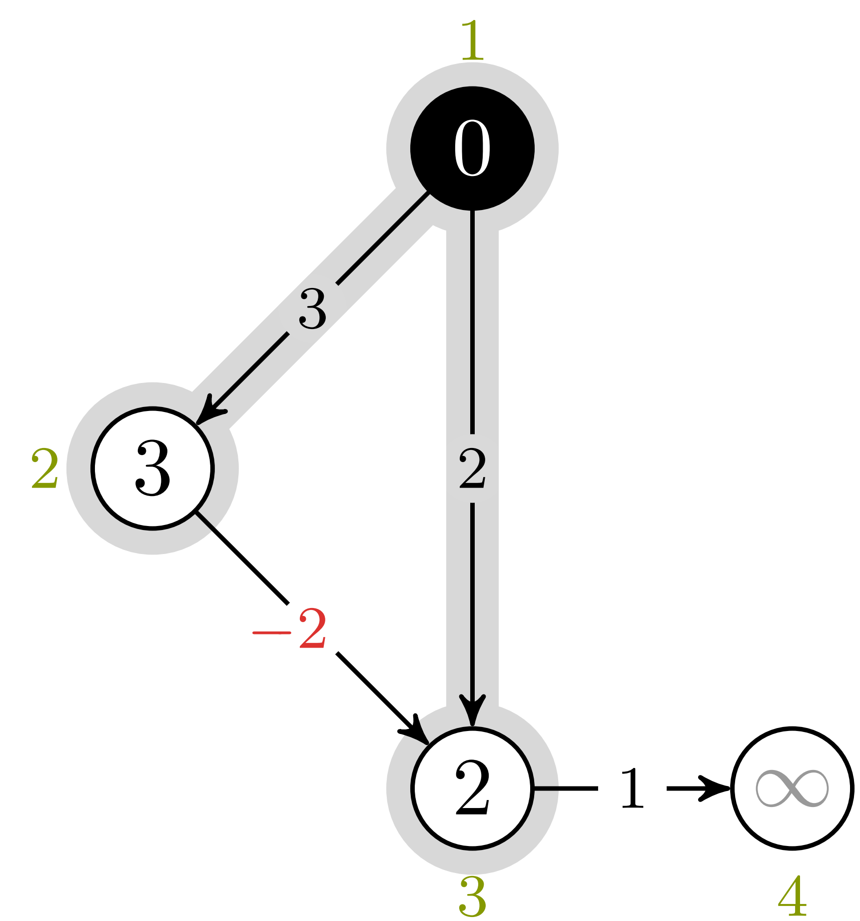
$u, v \;=\; \text{-}, \text{-}$

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



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

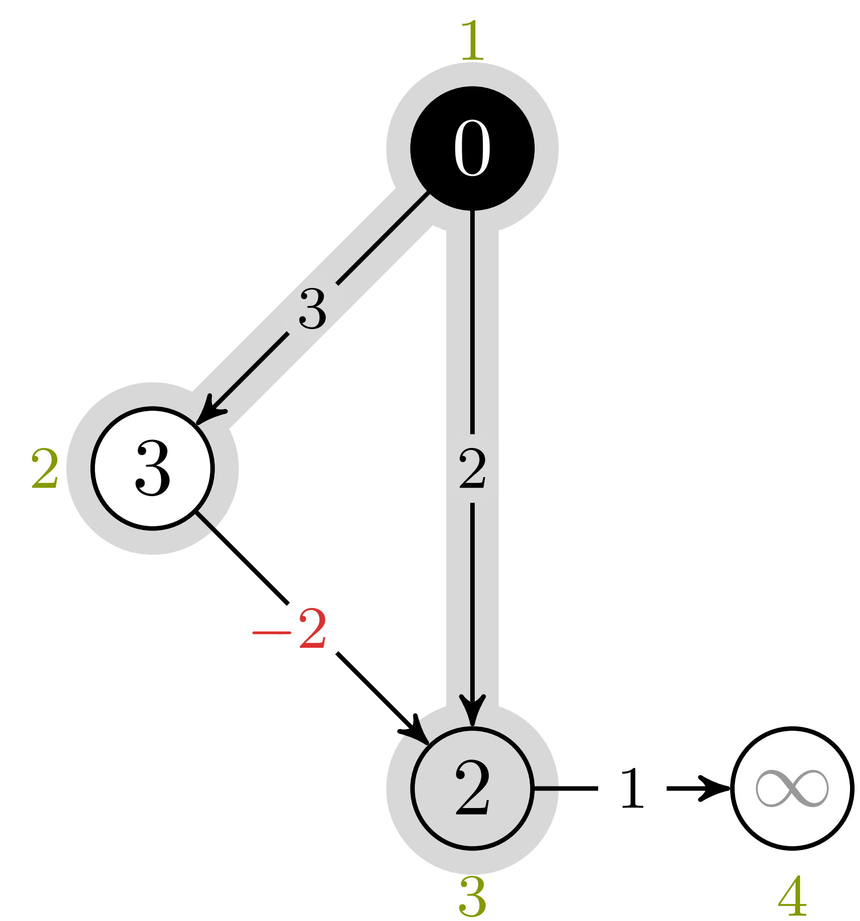
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```



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

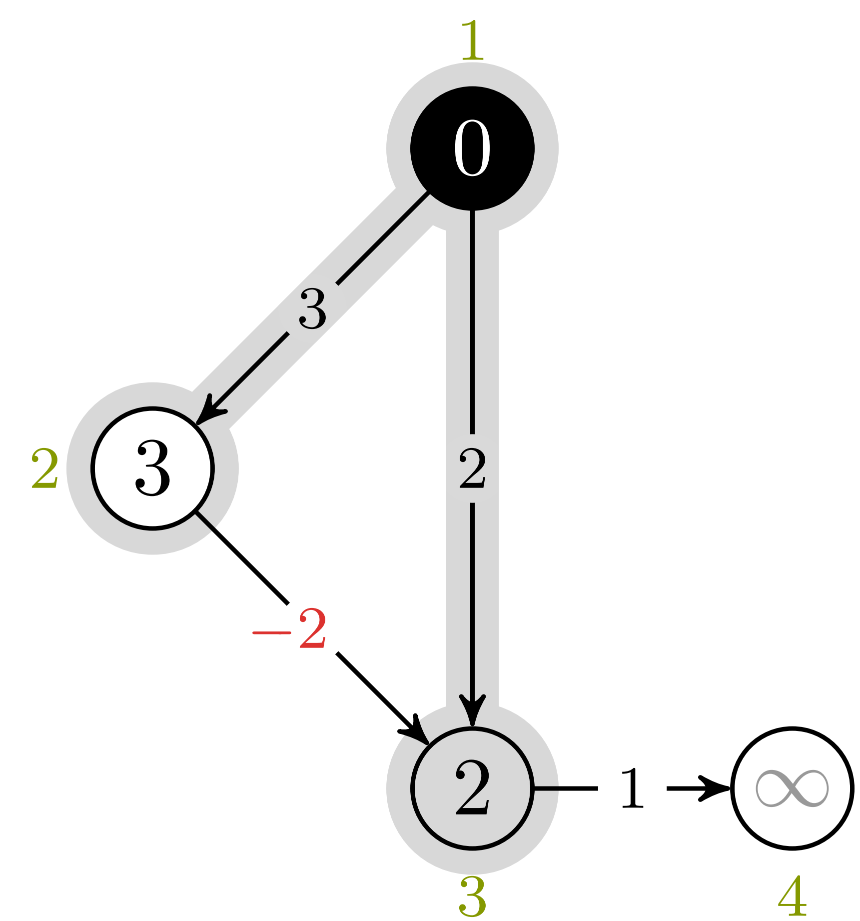
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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



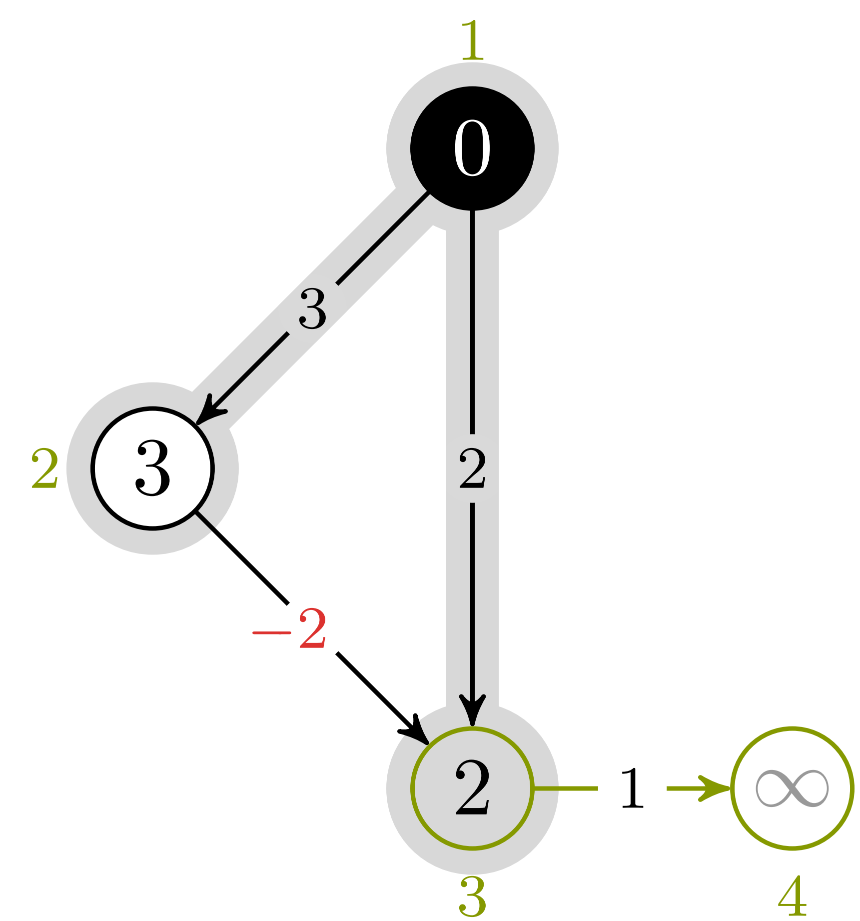
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
              RELAX(u, v, w)
```

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



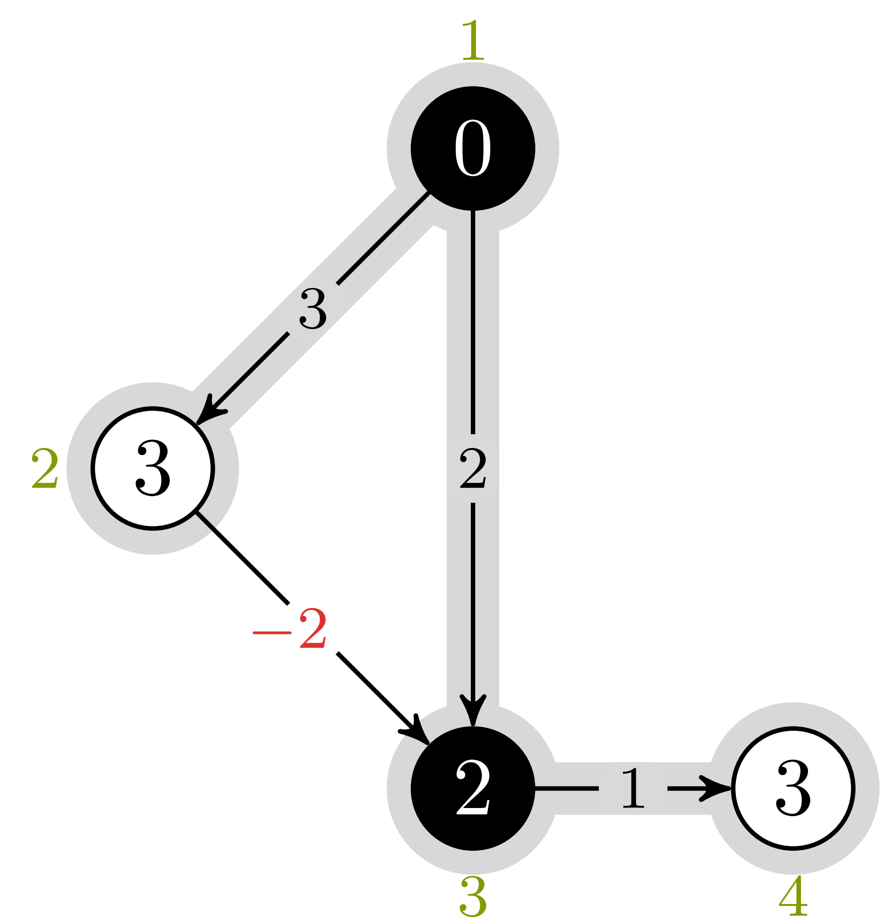
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
                RELAX(u, v, w)
```

$u, v \;=\; 3, 4$

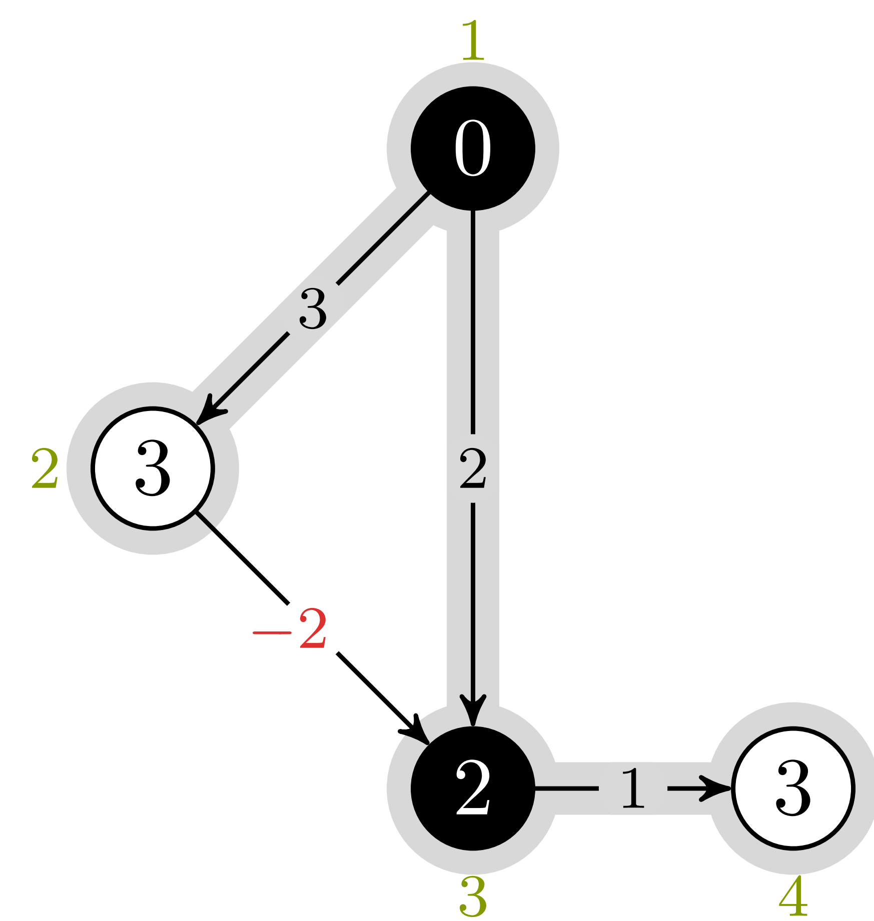


```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

$u, v = -, -$



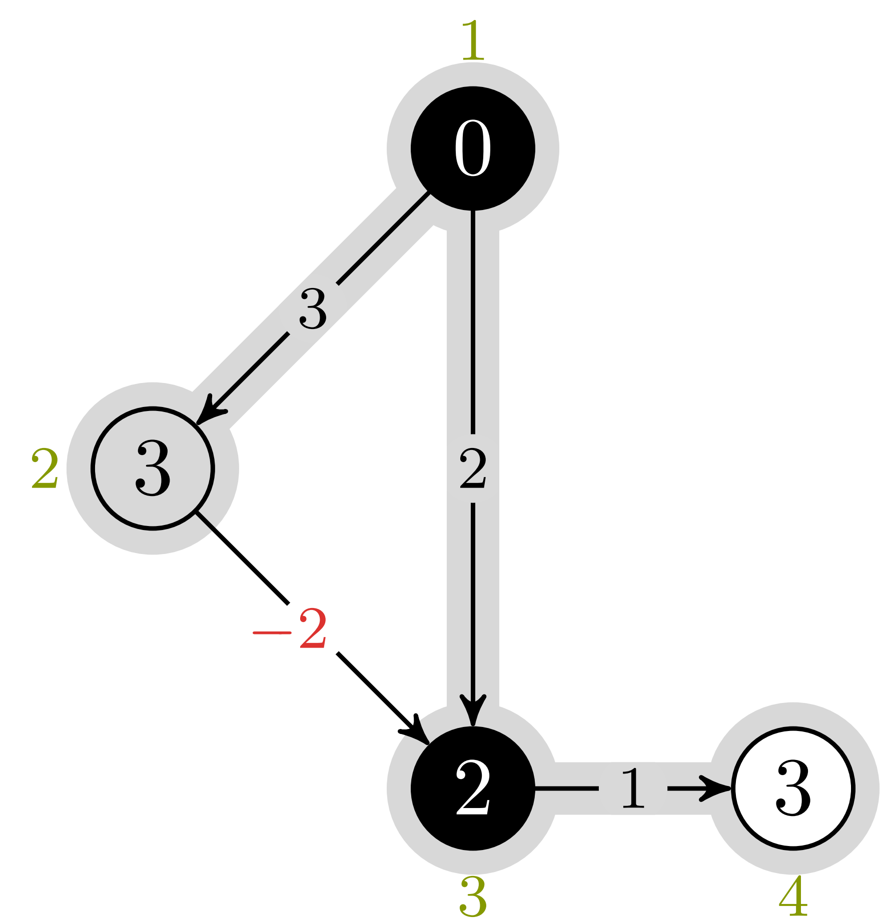
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



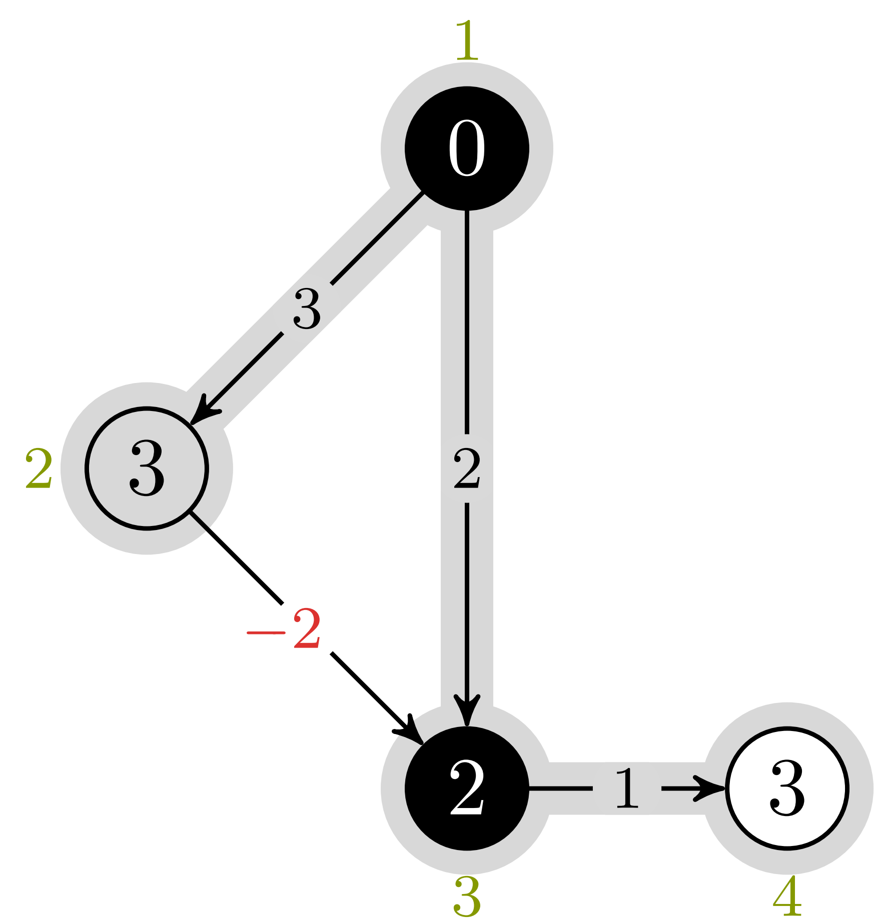
$u, v \;=\;$ -, -

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$u, v \;=\; 2, \text{-}$

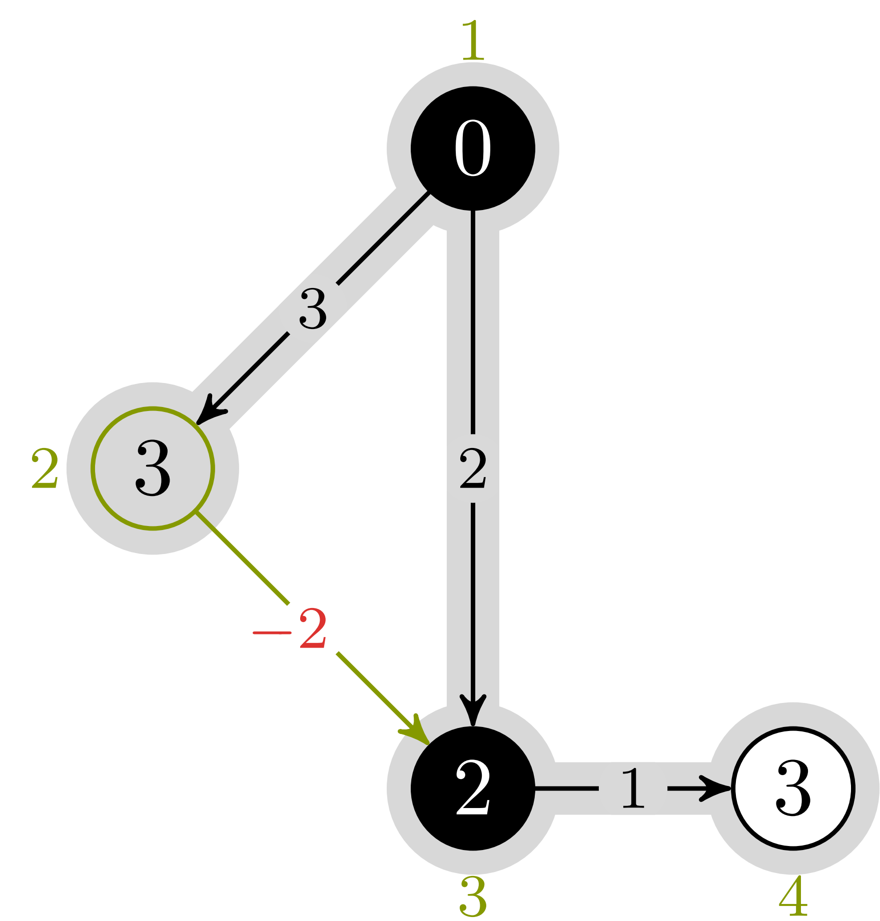


DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S = \emptyset$
3 $Q = G.V$
4 **while** $Q \neq \emptyset$
5 $u =$ EXTRACT-MIN(Q)
6 $S = S \cup \{u\}$
7 **for** each vertex $v \in G.Adj[u]$
8 RELAX(u, v, w)



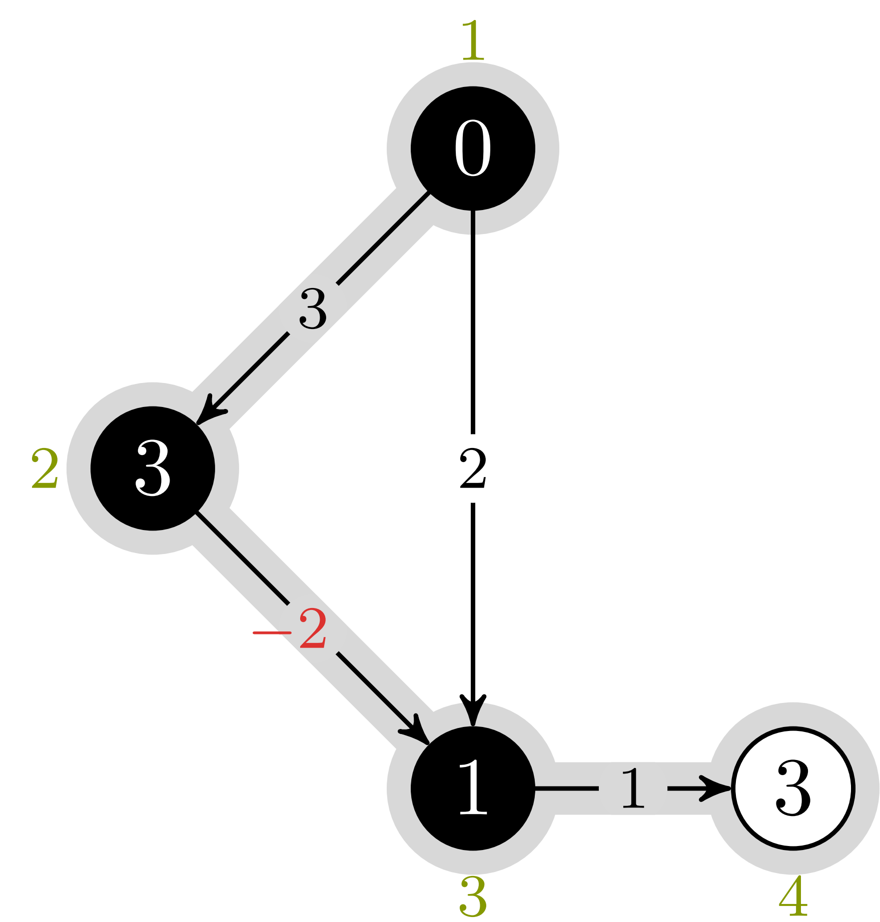
$u, v = 2, -$

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
              RELAX(u, v, w)
```



$u, v \;=\; 2, 3$

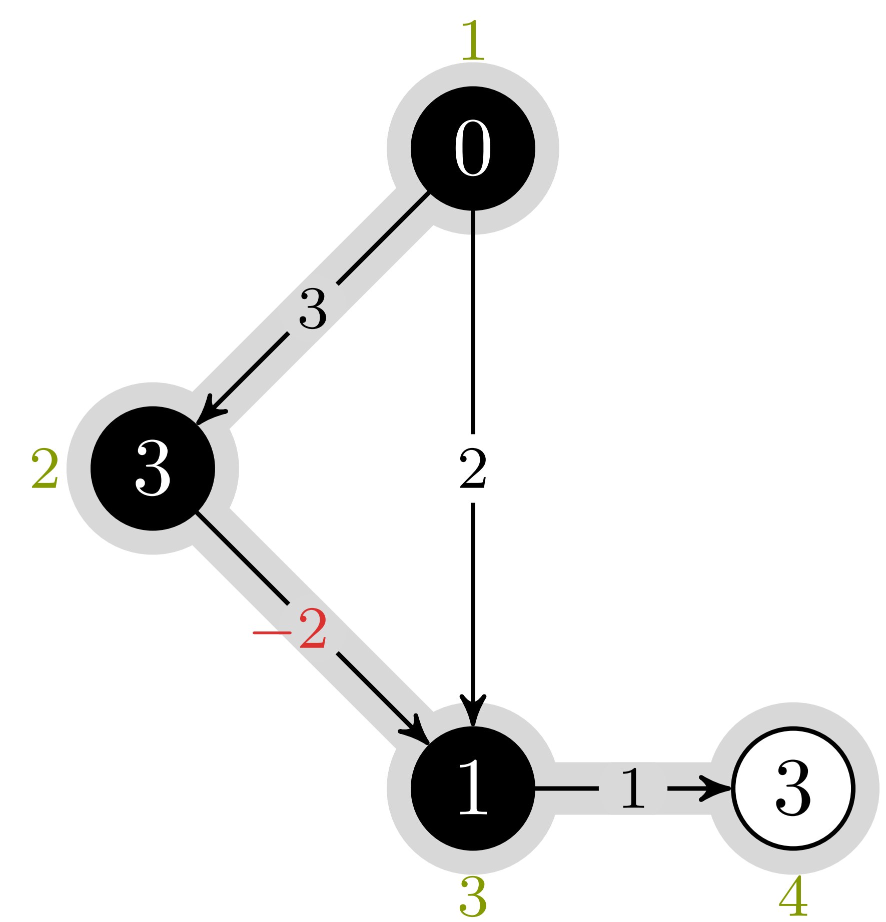
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    u = EXTRACT-MIN(Q)
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w)



u, v = -, -

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

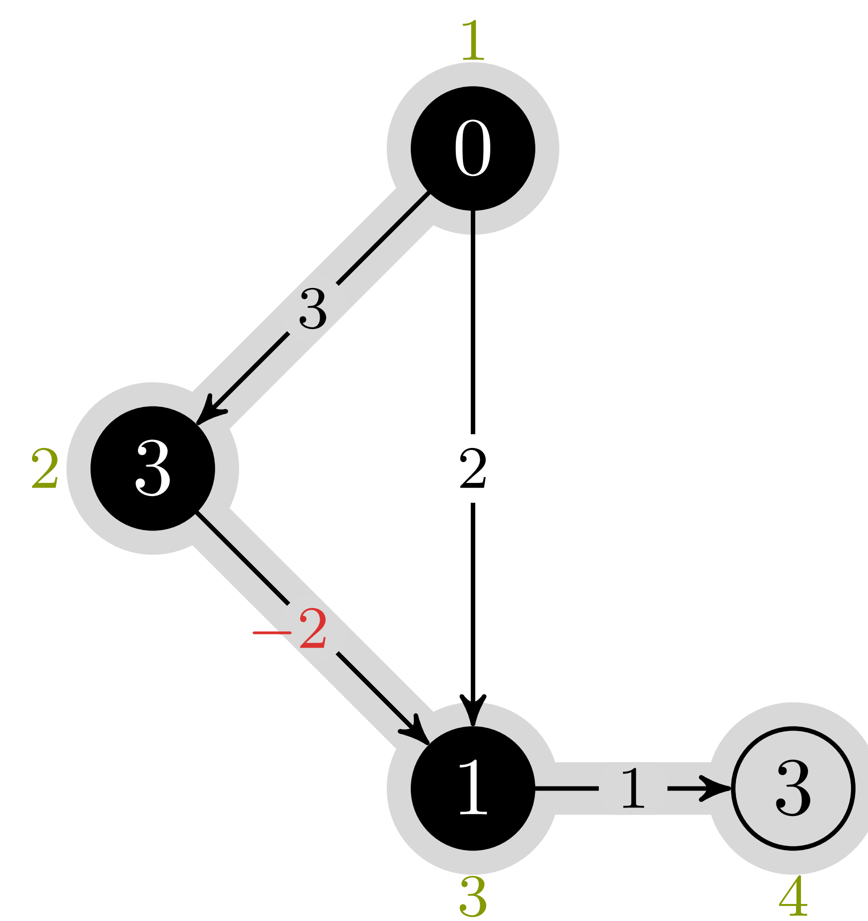
$u, v \; = \; \text{-}, \text{-}$



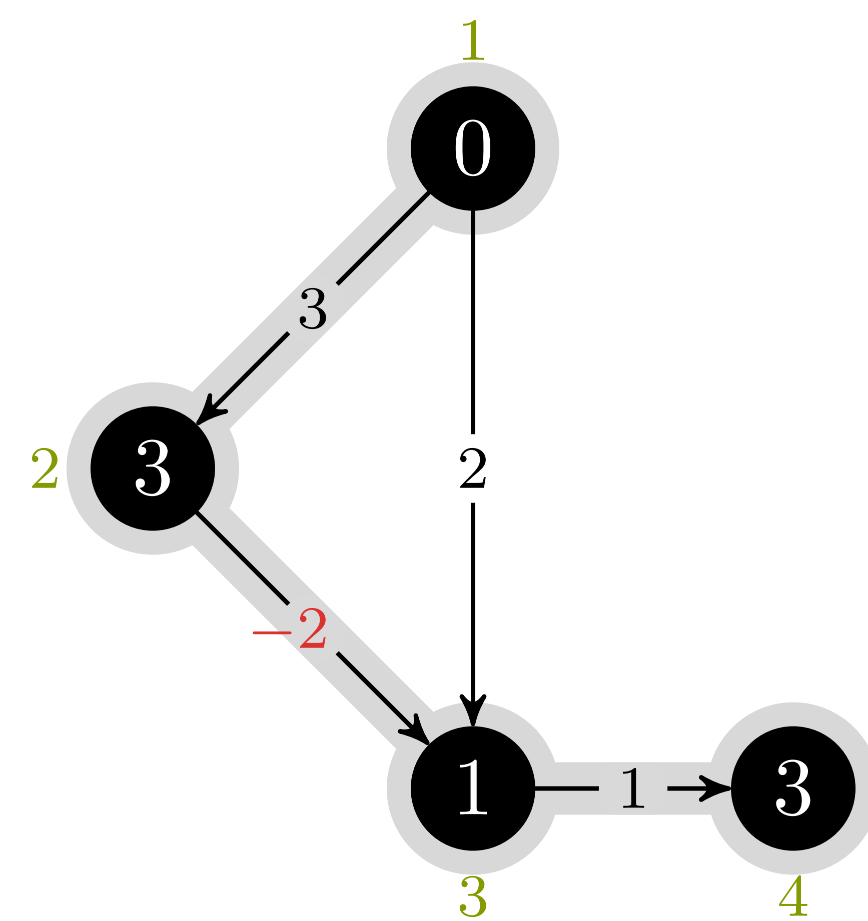
$\textsc{Dijkstra}(G, w, s)$

$\quad\textsc{Initialize-Single-Source}(G, s)$
$\quad S = \emptyset$
$\quad Q = G.V$
$\quad$ **while** $Q \neq \emptyset$
$\qquad u = \textsc{Extract-Min}(Q)$
$\qquad S = S \cup \{u\}$
$\qquad$ **for** each vertex $v \in G.Adj[u]$
$\qquad\quad \textsc{Relax}(u, v, w)$

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



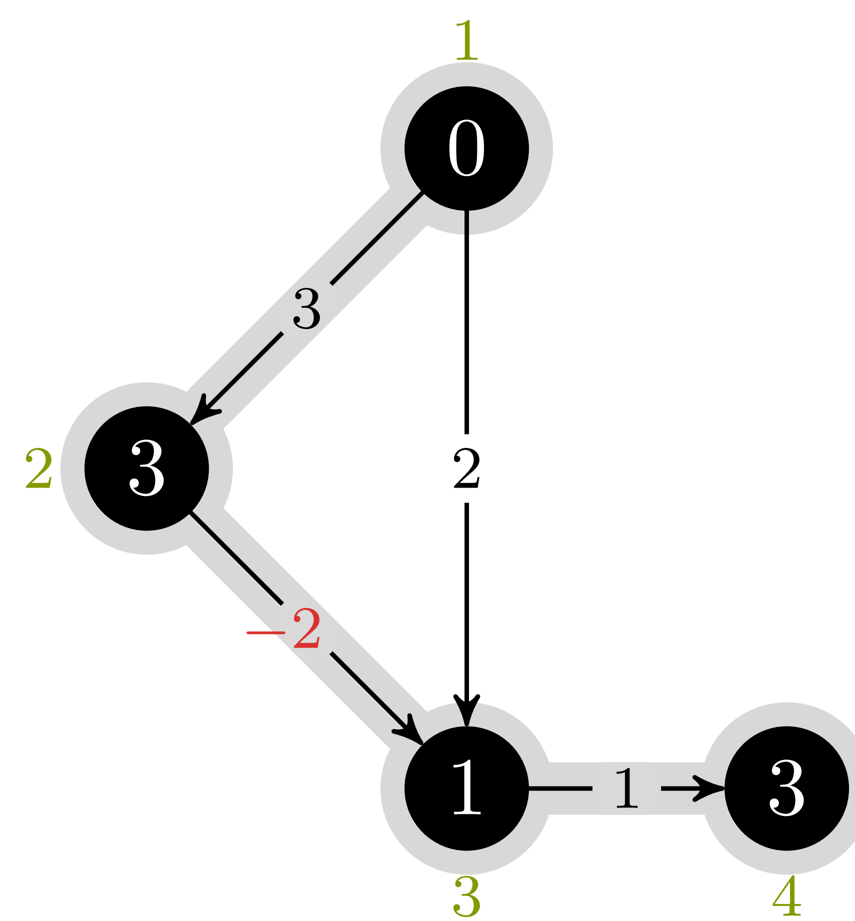
DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S = ∅
3 Q = G.V
4 **while** Q ≠ ∅
5 u = EXTRACT-MIN(Q)
6 S = S ∪ {u}
7 **for** each vertex $v \in$ G.$Adj[u]$
8 RELAX(u, v, w)



$u, v = -, -$

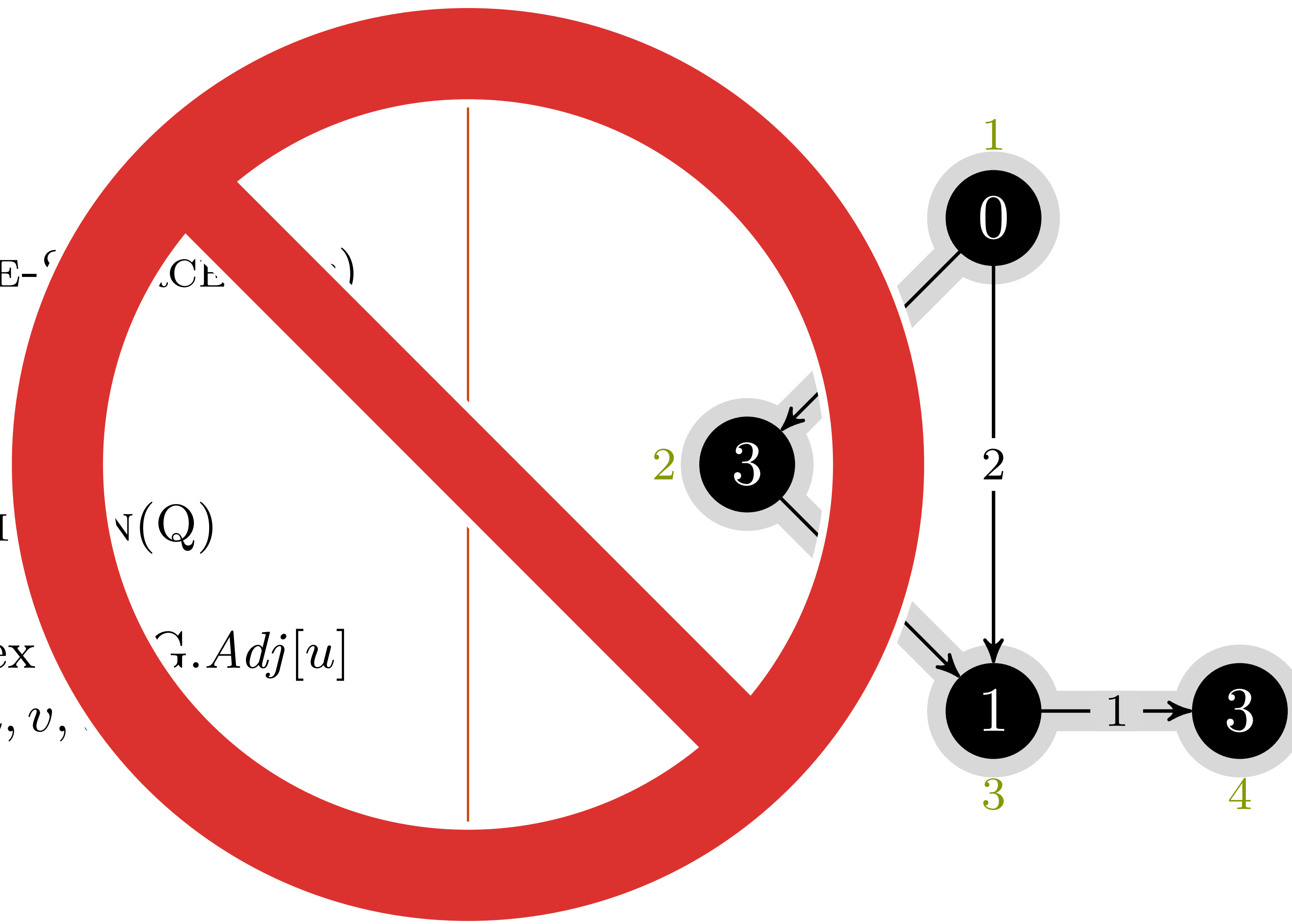
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
              RELAX(u, v, w)
```

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



3.d er nå feil

```
Dijkstra(G, w, s)
Initialize-Single-S      ce     )
S = ∅
Q = G.V
while Q ≠ ∅
      u = Extract      n(Q)
      S = S ∪ {u}
      for each vertex      G.Adj[u]
            Relax(u, v,
```



$u, v = -, -$

Negative kanter forbudt! $B.d$ er nå feil

Dijkstra › Kjøretid

Operasjon	Antall	Kjøretid

Operasjon	Antall	Kjøretid
Initialisering		

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
Build-Heap		

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
Build-Heap	1	

Operasjon	Antall	Kjøretid
Initialisering	1	$\Theta(V)$
Build-Heap	1	$\Theta(V)$

Operasjon	Antall	Kjøretid
Initialisering	1	$\Theta(V)$
Build-Heap	1	$\Theta(V)$
Extract-Min		

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
BUILD-HEAP	1	$\Theta(V)$
EXTRACT-MIN	V	

Operasjon	**Antall**	**Kjøretid**
Initialisering	1	$\Theta(V)$
Build-Heap	1	$\Theta(V)$
Extract-Min	V	$O(\lg V)$

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

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

*Nødvendig i RELAX

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

*Nødvendig i Relax

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

*Nødvendig i Relax

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

Totalt: $O(V \lg V + E \lg V)$

*Nødvendig i RELAX

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

Totalt: $O(V \lg V + E \lg V)$

*Nødvendig i RELAX

Med binærheap; bedre enn lineært søk for $E = o(V^2/\lg V)$

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

Totalt: $O(V \lg V + E \lg V)$

*Nødvendig i RELAX

Boka bruker $V \times$ INSERT; fortsatt $O(E \lg V)$

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

Totalt: $O(V \lg V + E \lg V)$

*Nødvendig i RELAX

Fibonacci-heaps: DECREASE-KEY er $O(1)$; vi får $O(V \lg V + E)$

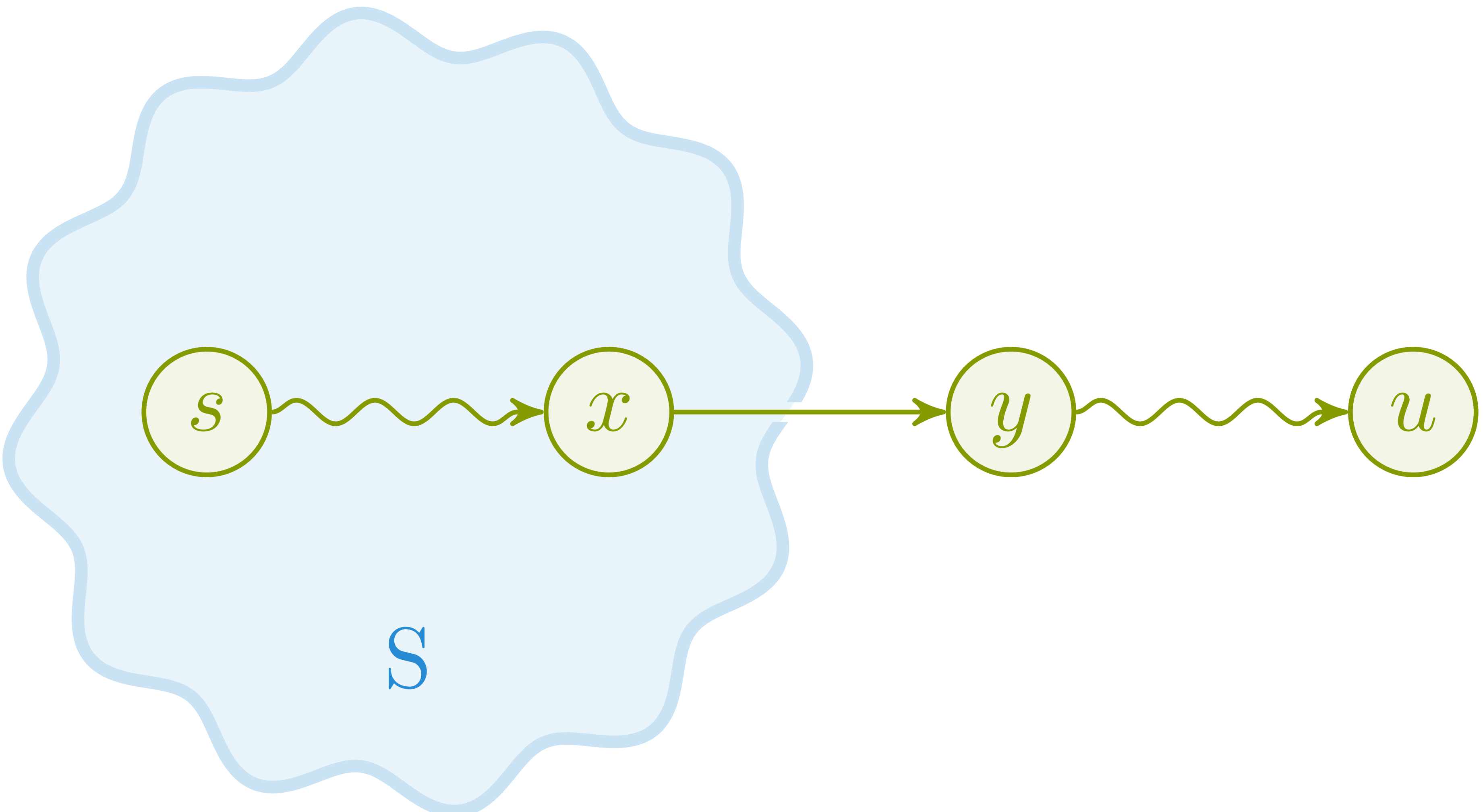
1. **Dekomponering**
2. **DAG-Shortest-Path**
3. **Kantslakking**
4. **Bellman-Ford**
5. **Dijkstras algoritme**

Bonusmateriale

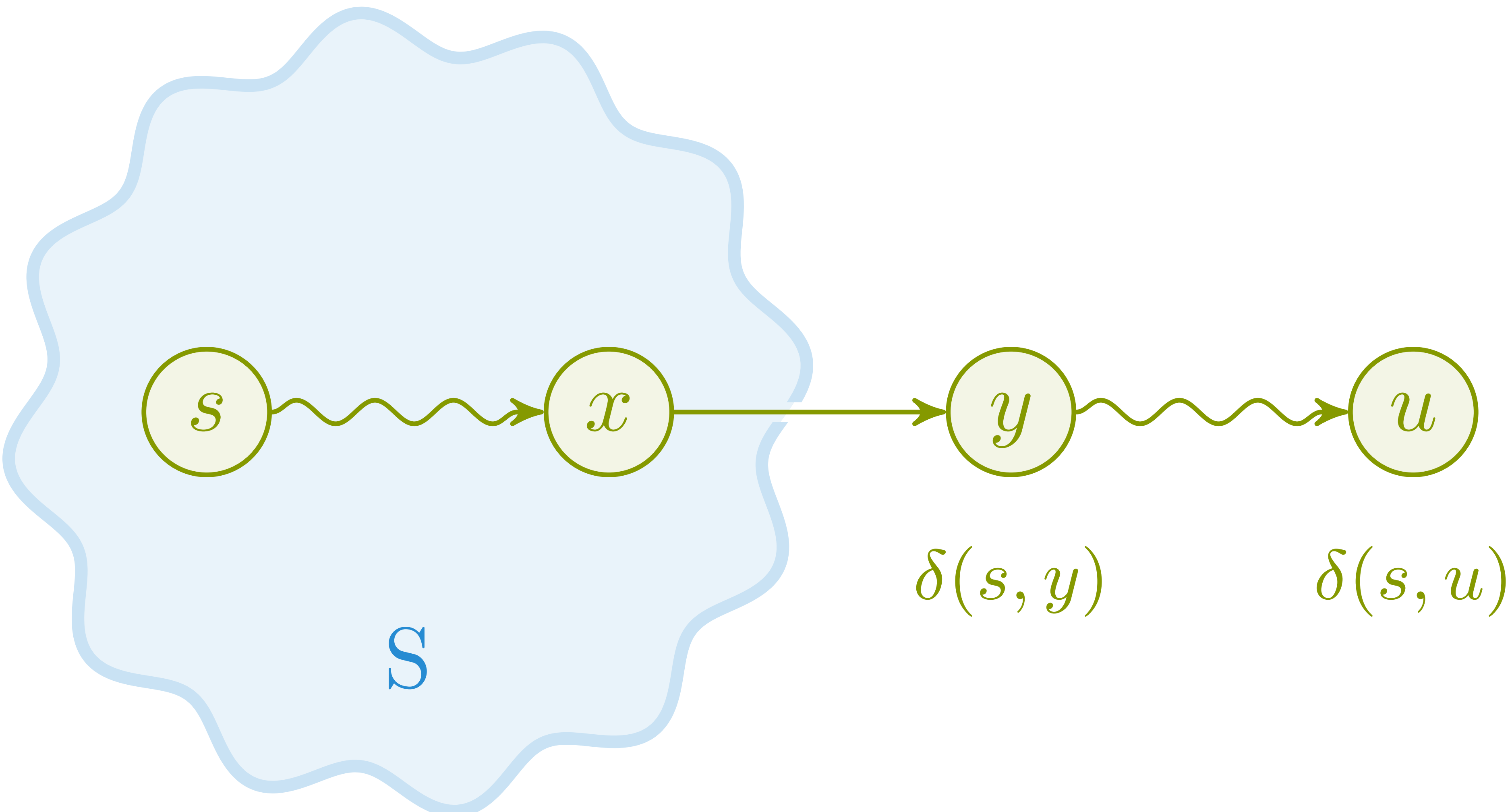
Korteste vei › Dijkstra › Korrekthet

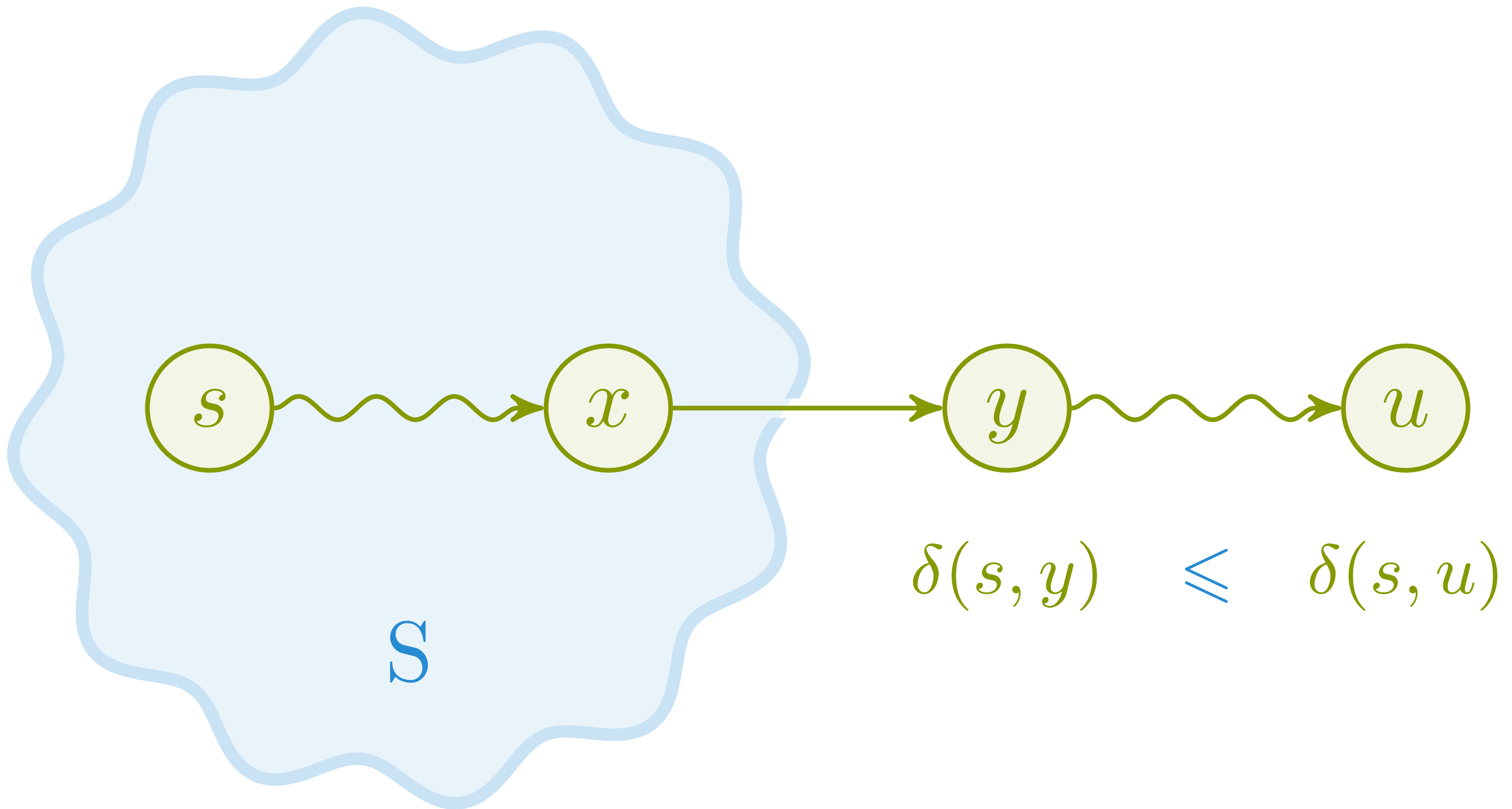
La u være den neste som skal besøkes; kan $u.d$ være feil?

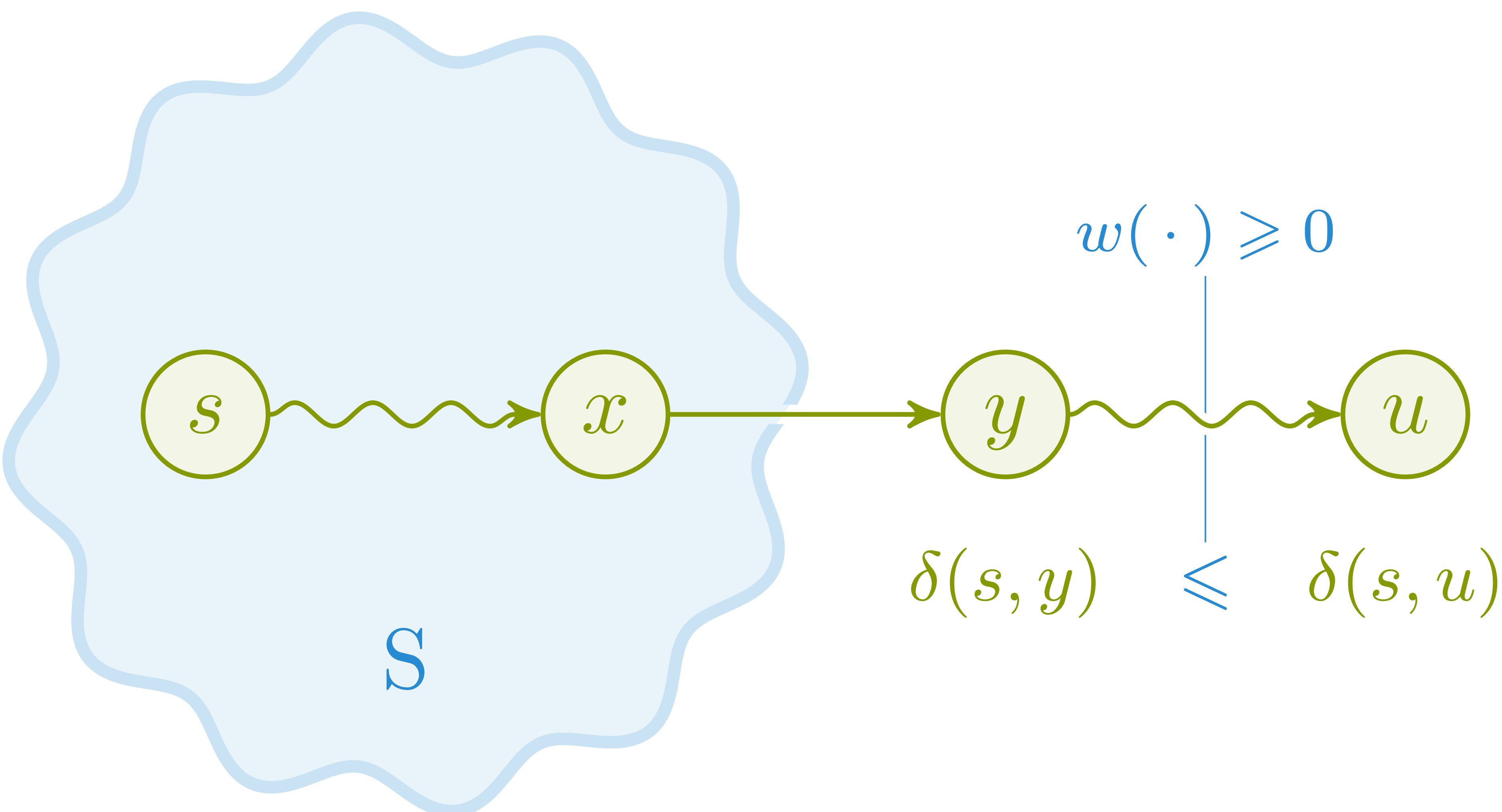
(x, y) ligger på en av de korteste stiene fra s til u



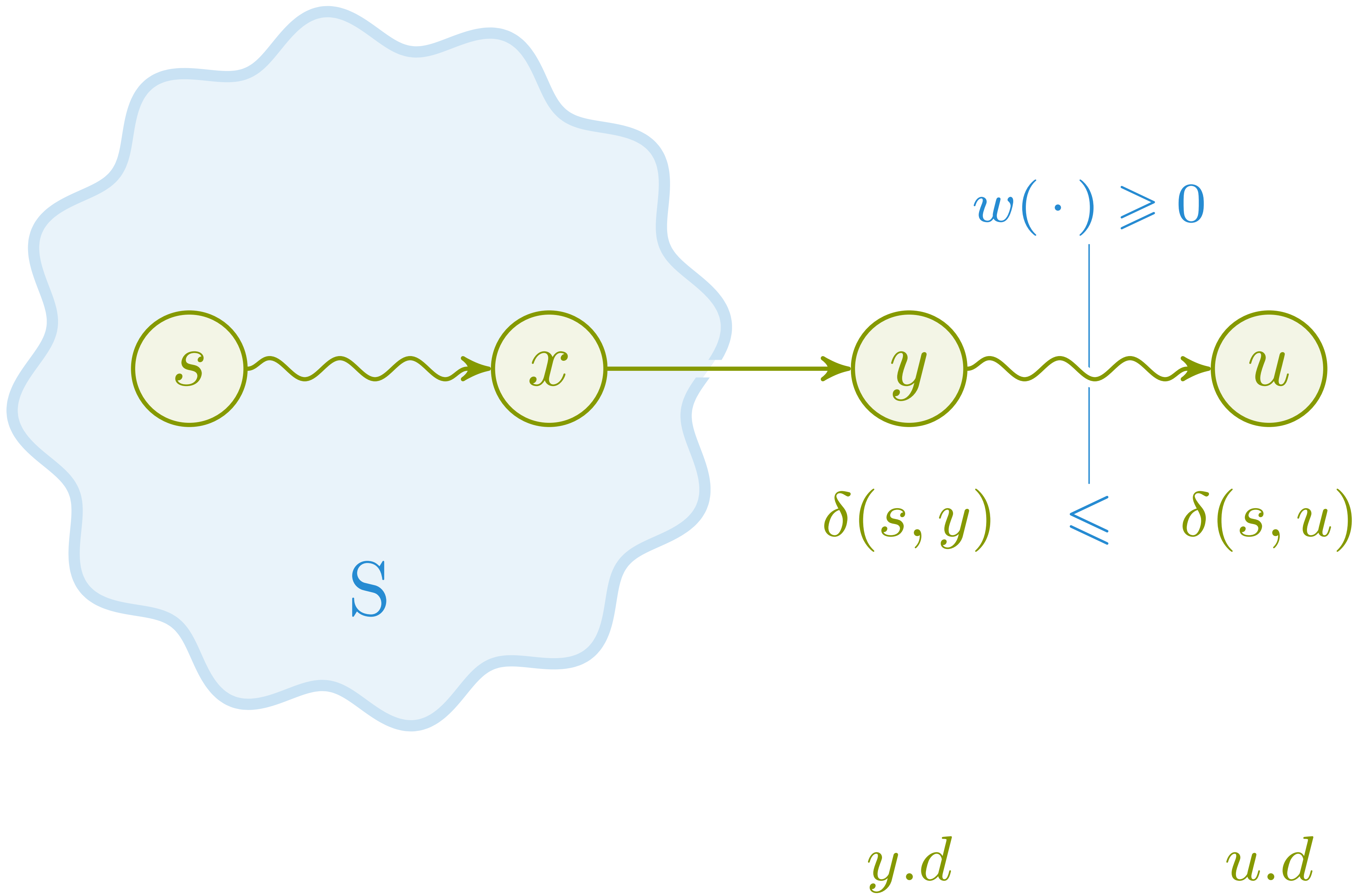
x ligger i S; vi kan ha $s = x$ eller $y = u$; $y \rightsquigarrow u$ kan gå innom S

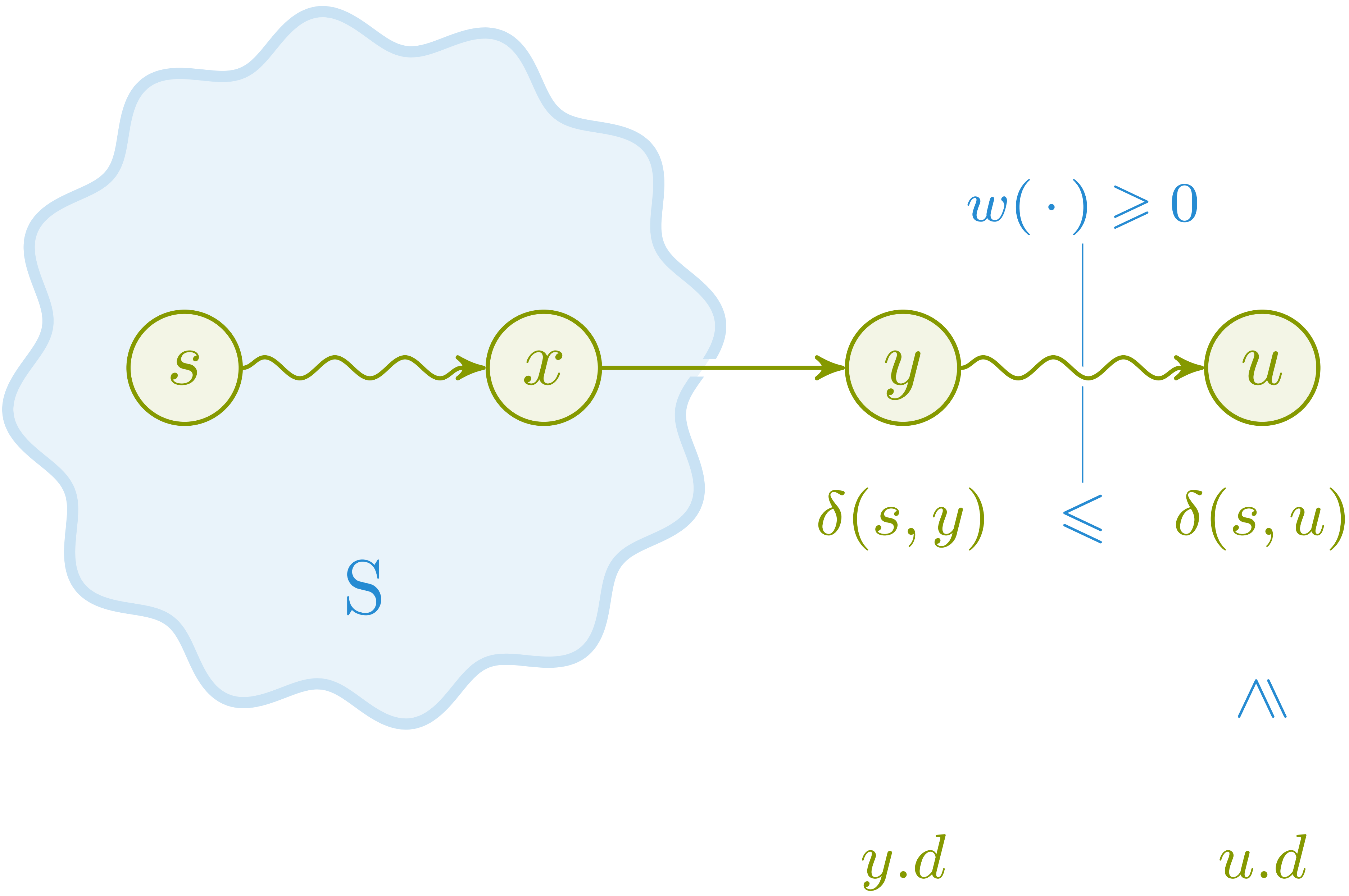

s
x
y
u
$\delta(s, y)$
$\delta(s, u)$
S

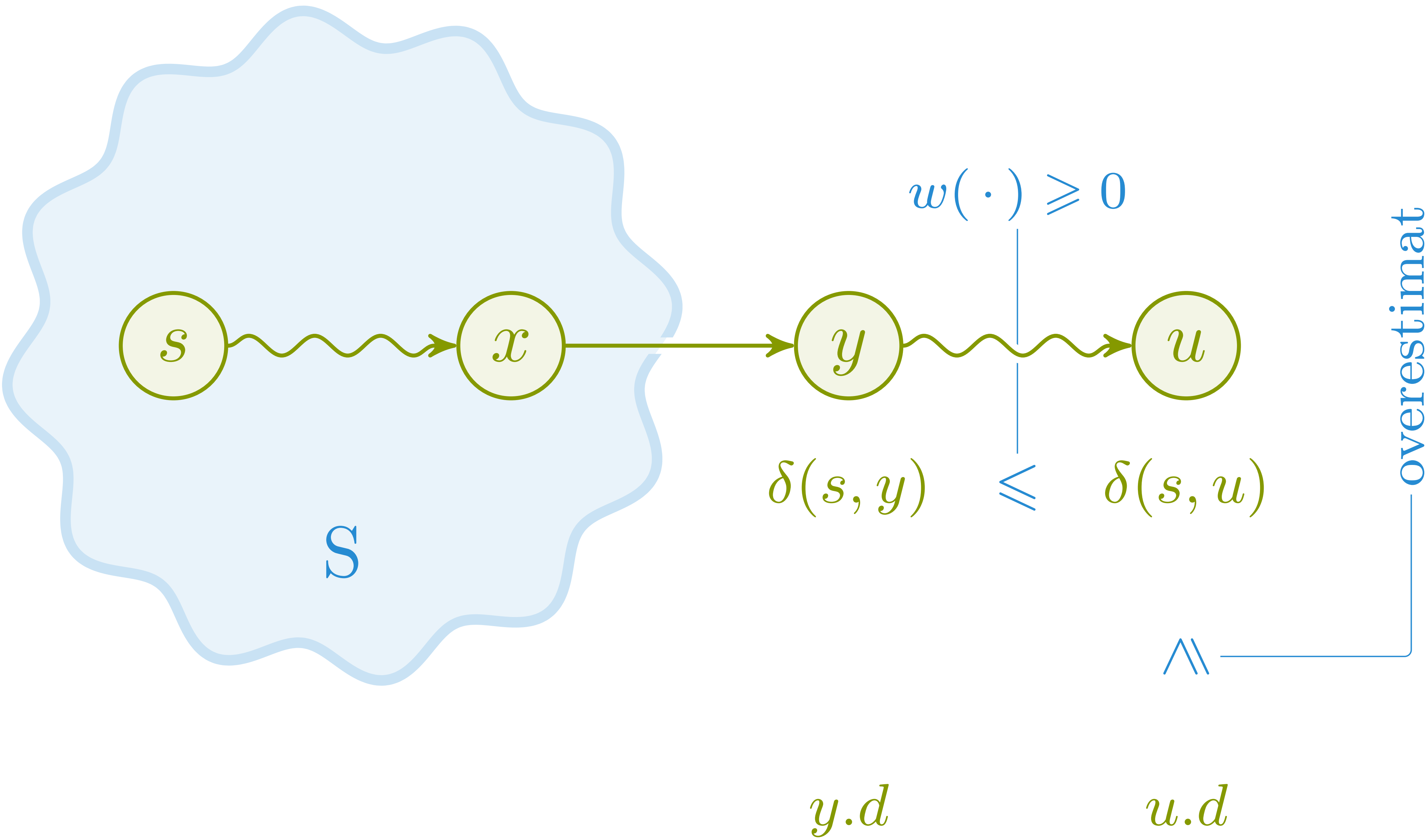

s
x
y
u
$\delta(s, y) \leqslant \delta(s, u)$
S

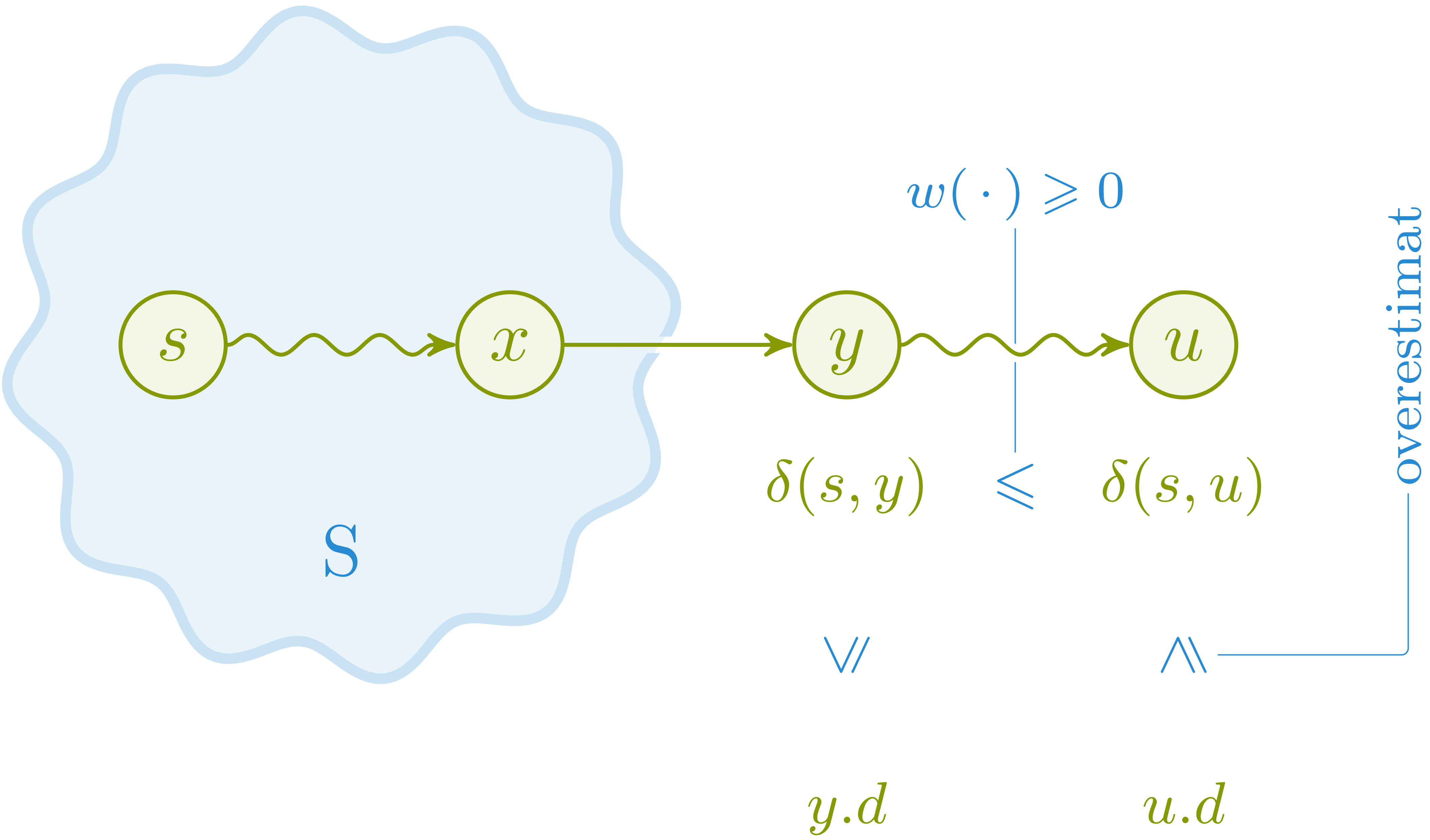


Ikke-negative vekter $\implies$ avstandene synker ikke

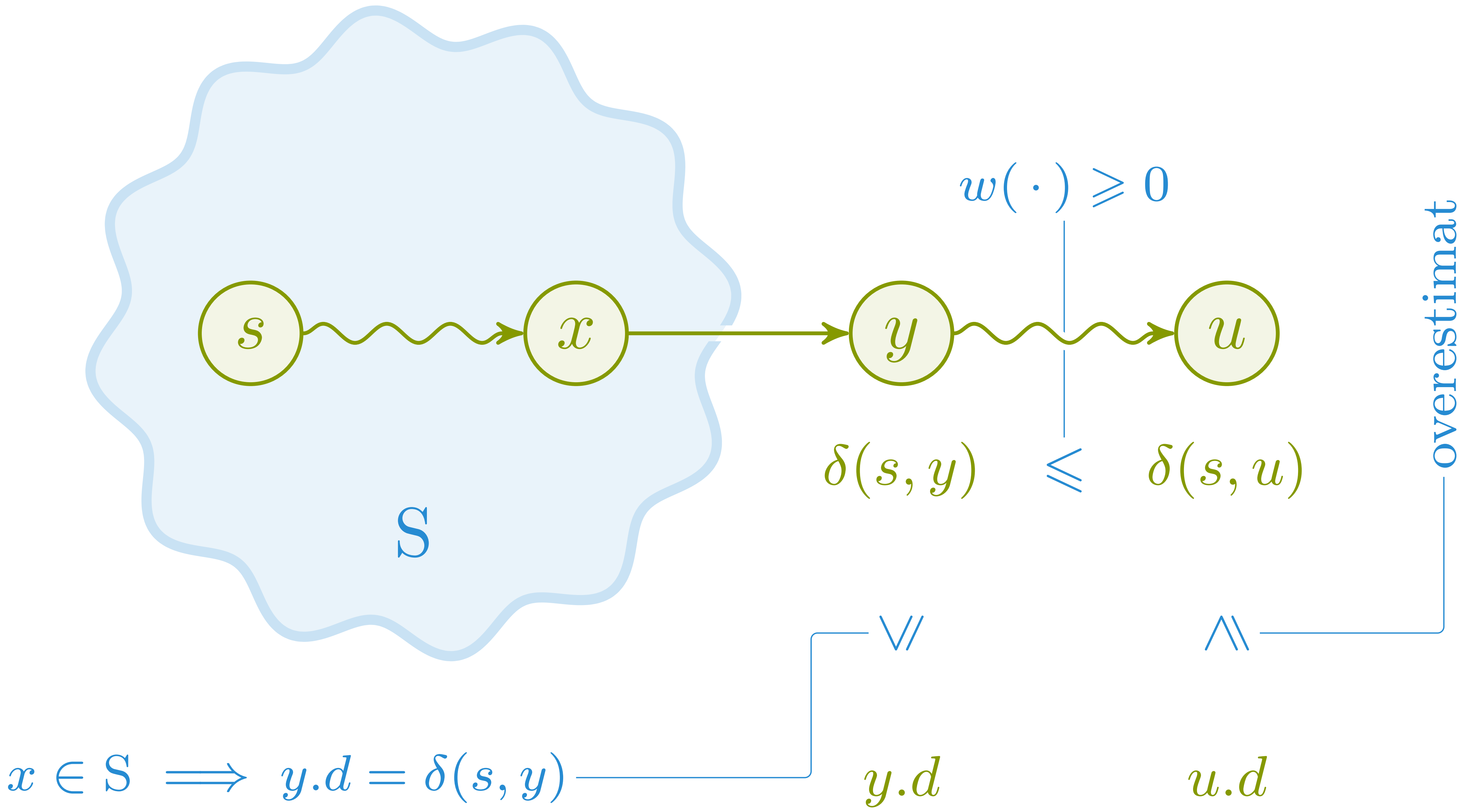

s
x
y
u
$w(\cdot) \geqslant 0$
$\delta(s,y)$
$\leqslant$
$\delta(s,u)$
S
$y.d$
$u.d$

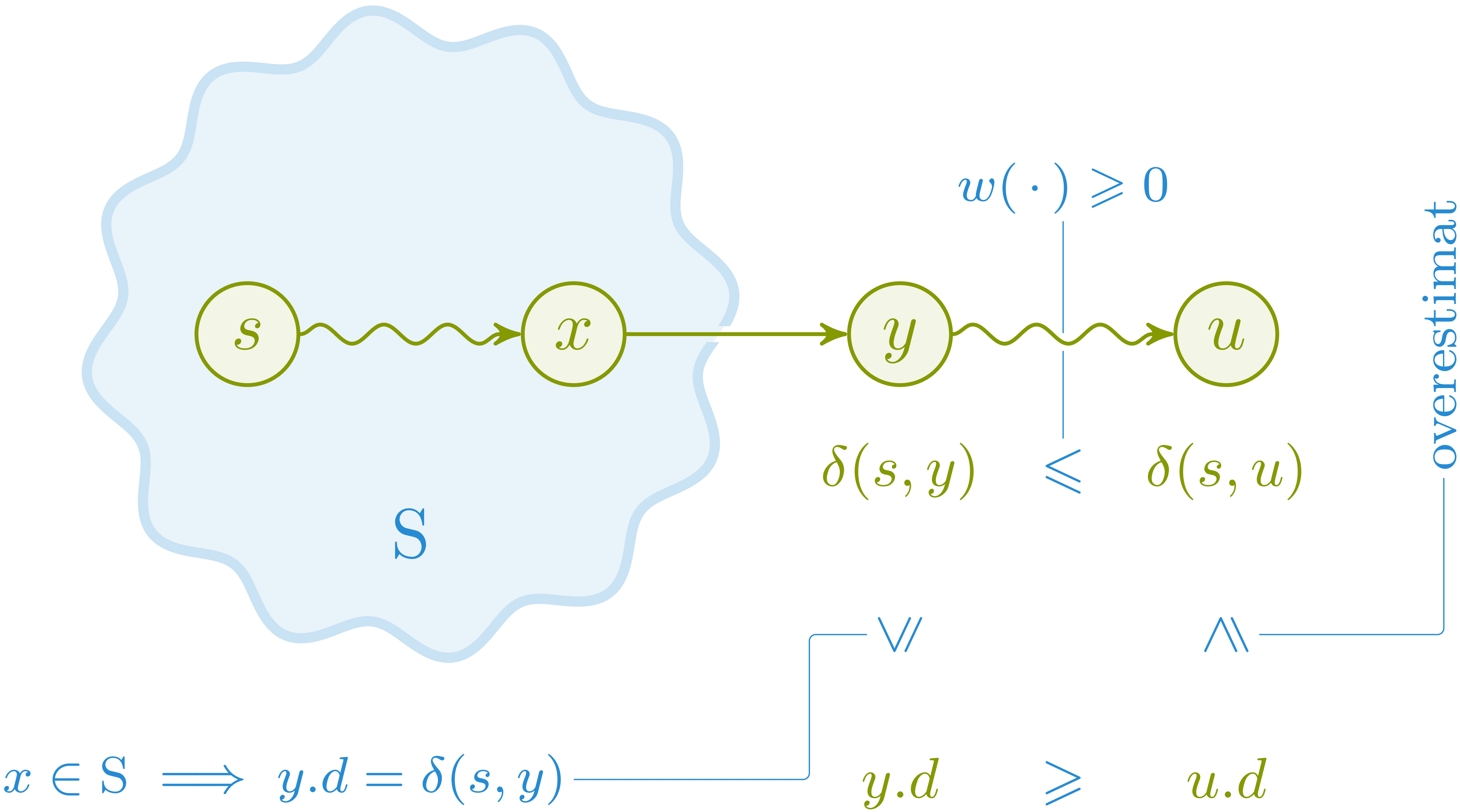

s
x
y
u
S
w(·) ⩾ 0
δ(s, y)
⩽
δ(s, u)
⩽
y.d
u.d


s
x
y
u
S
$w(\cdot) \geqslant 0$
$\delta(s, y) \leqslant \delta(s, u)$
overestimat
$\wedge\!\!\!\|$
$y.d$
$u.d$

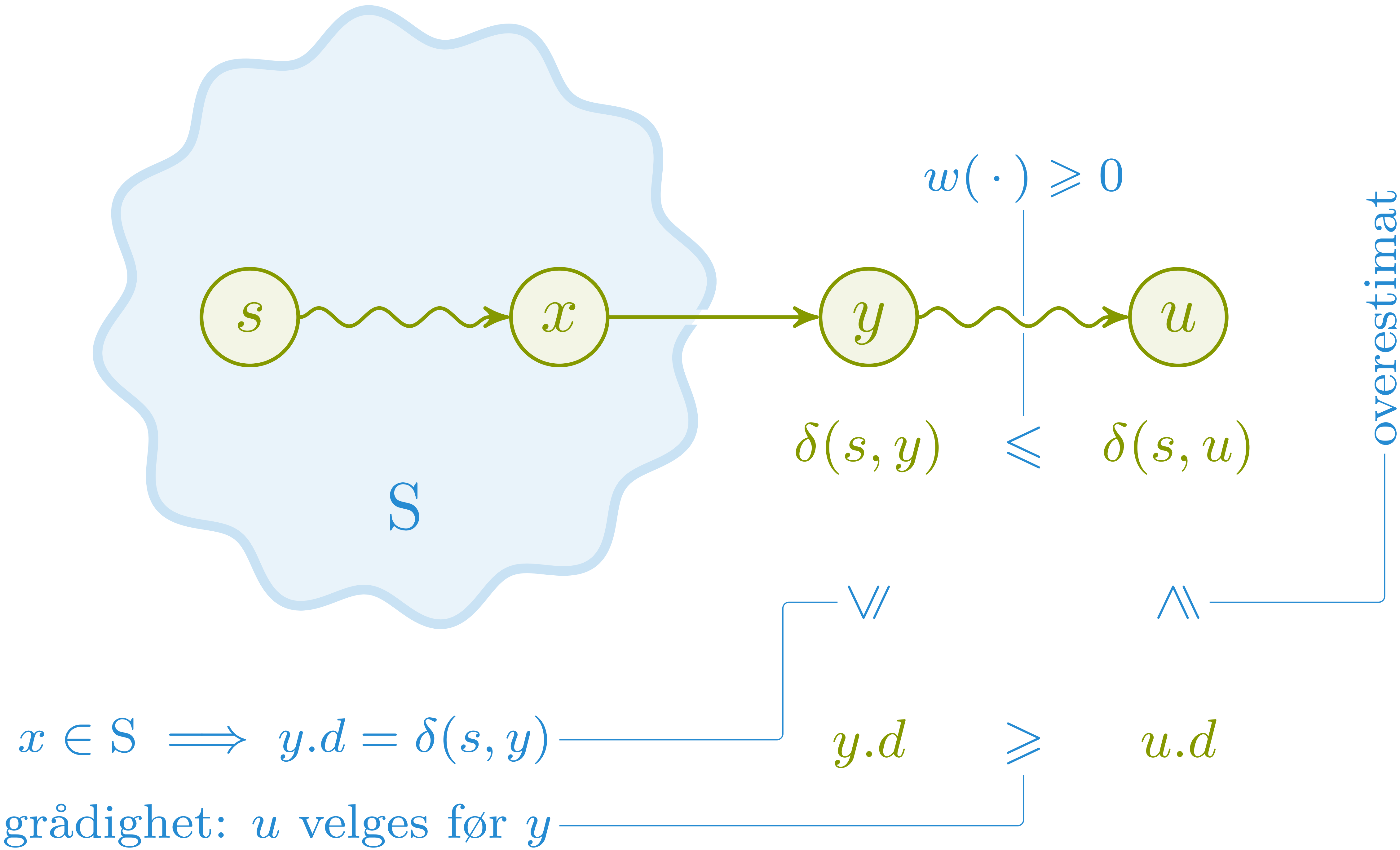


s
x
y
u
S
$w(\,\cdot\,) \geqslant 0$
$\delta(s,y) \leqslant \delta(s,u)$
$\vee\!\!/\!/$
$\wedge\!\!\backslash\!\backslash$
$y.d$
$u.d$
overestimat

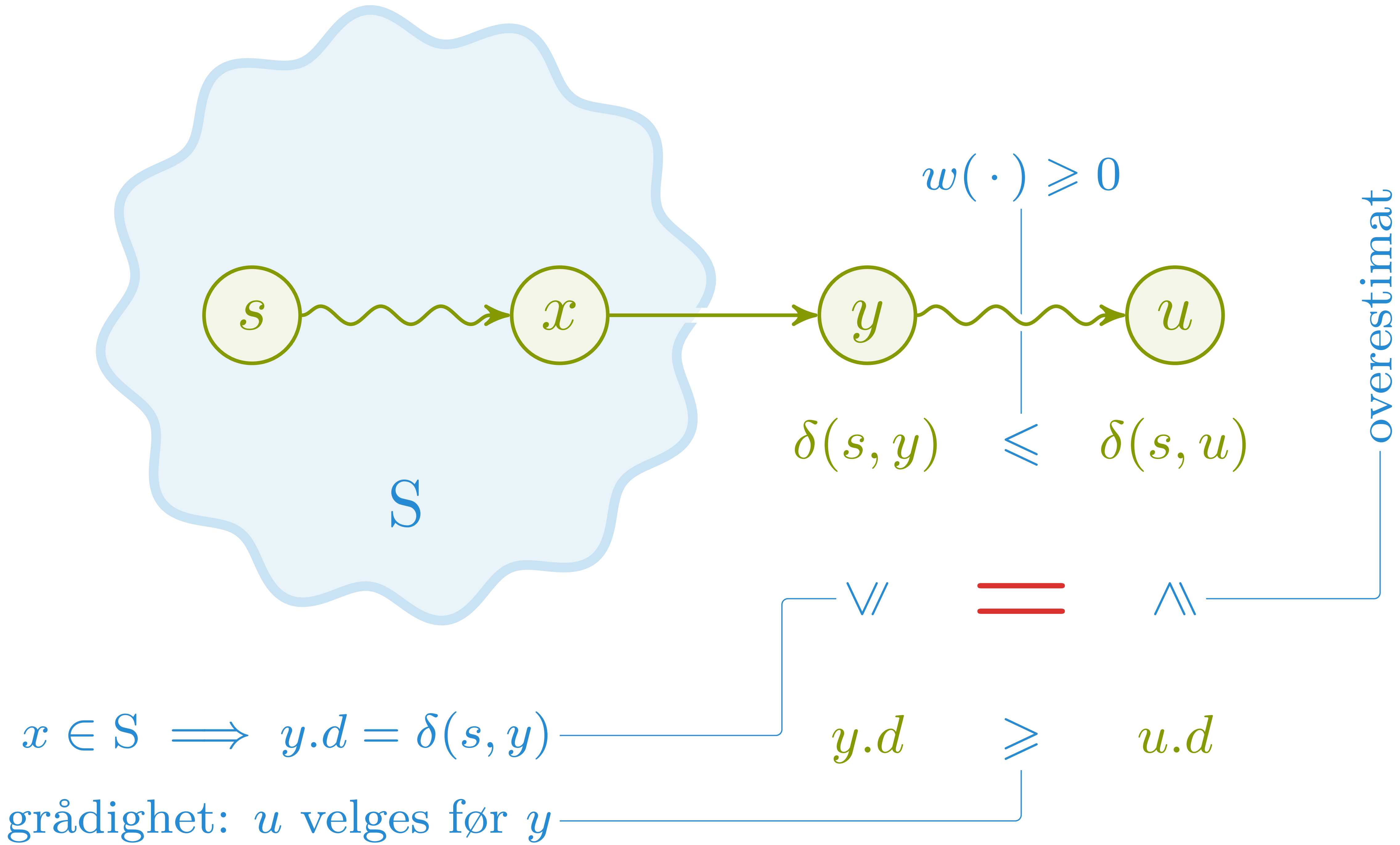

s
x
y
u
S
$w(\cdot) \geqslant 0$
$\delta(s, y) \leqslant \delta(s, u)$
overestimat
$\vee\!/$
$\wedge\!\backslash$
$x \in S \implies y.d = \delta(s, y)$
$y.d$
$u.d$



s
x
y
u
S
$w(\,\cdot\,) \geqslant 0$
overestimat
$\delta(s,y) \leqslant \delta(s,u)$
$\vee\!/\!\!$ $\wedge\!\backslash\!\!$
$x \in S \implies y.d = \delta(s,y)$
$y.d \geqslant u.d$



s
x
y
u
S
$w(\cdot) \geqslant 0$
overestimat
$\delta(s, y) \leqslant \delta(s, u)$
$\vee\!\!/ \quad \wedge\!\!\backslash$
$y.d \geqslant u.d$
$x \in \mathrm{S} \implies y.d = \delta(s, y)$
grådighet: u velges før y


s
x
y
u
S
$w(\,\cdot\,) \geqslant 0$
overestimat
$\delta(s, y) \leqslant \delta(s, u)$
$\veebar\!\!\!/ \quad = \quad \wedge\!\!\!\backslash$
$x \in S \implies y.d = \delta(s, y)$
$y.d \geqslant u.d$
grådighet: u velges før y

$$\begin{array}{ccc}
\delta(s,y) & \leqslant & \delta(s,u) \\
\rotatebox{90}{$\leqslant$} & = & \rotatebox{90}{$\geqslant$} \\
y.d & \geqslant & u.d
\end{array}$$

Når u velges, så er $u.d$ lik $\delta(s,u)$, altså korrekt