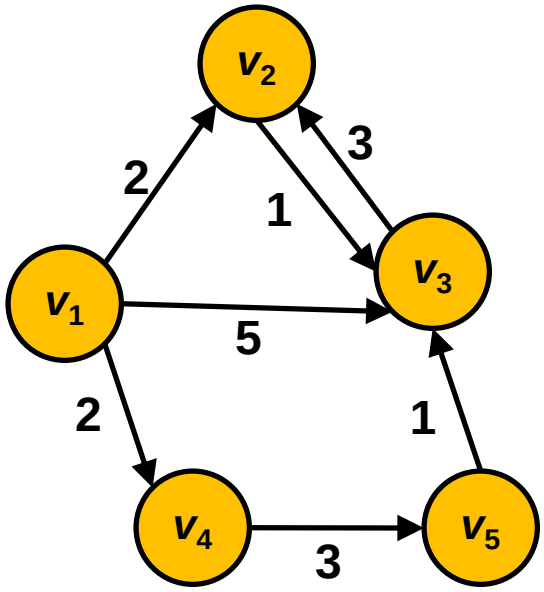
Grundlæggende Algoritmer og Datastrukturer

Korteste Veje – mellem alle par af knuder
[CLRS, kapitel 25]

Korteste Veje mellem alle Par af Knude

Hvis alle kanter har positive vægte – kør Dijkstra's algoritme n gange, én gang for hvert v_i som startknude

$O(n \cdot m \cdot \log n)$



d_{ij}

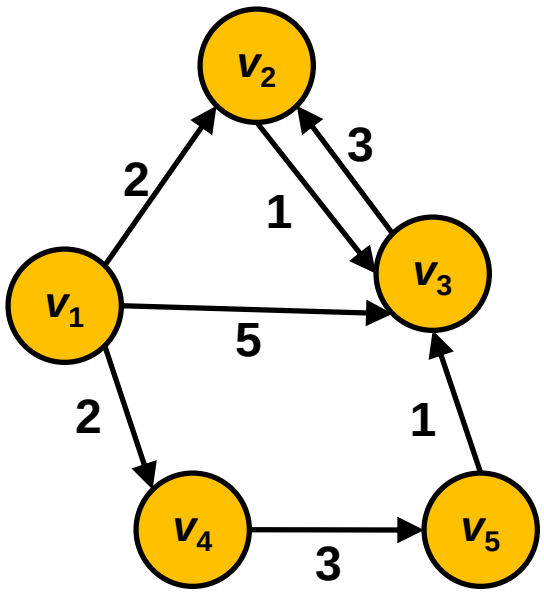
	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL		5	NIL	4
5	NIL	3	5	NIL	NIL

$\pi_{4,2}$?

a) 1
b) 2
c) 3
d) 4
e) 5
f) Ved ikke



d_{ij}

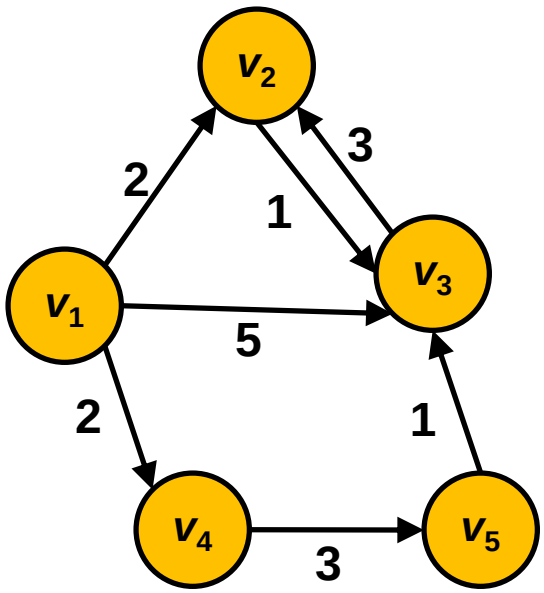
	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL		5	NIL	4
5	NIL	3	5	NIL	NIL

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
1  if i == j
2        print i
3  elseif π_ij == NIL
4        print "no path from" i "to" j "exists"
5  else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
6        print j
```

Tid $O(n)$



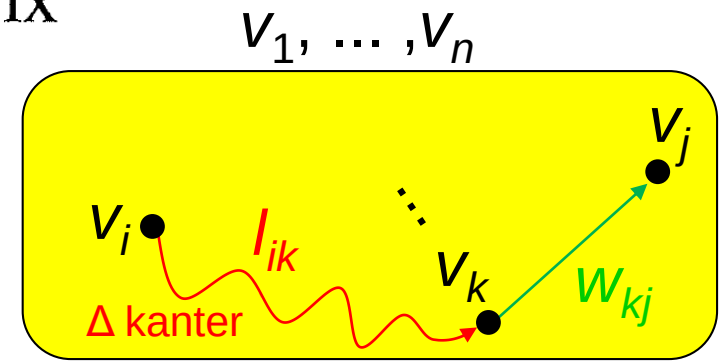
d_{ij}

	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL	3	5	NIL	4
5	NIL	3	5	NIL	NIL

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

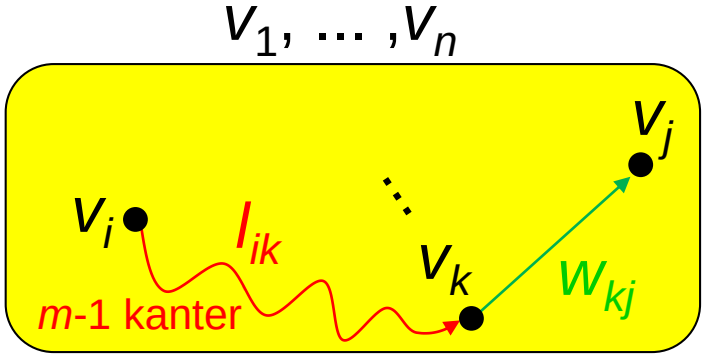


L_{ij} = korteste afstand fra *i* til *j* for stier med Δ kanter
W = vægtmatricen
L'_{ij} = korteste afstand fra *i* til *j* for stier med Δ+1 kanter

Tid $O(n^3)$

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

Tid $O(n^3)$



SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

```
1  n = W.rows
2  L^(1) = W
3  for m = 2 to n − 1
4      let L^(m) be a new n × n matrix
5      L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  return L^(n−1)
```

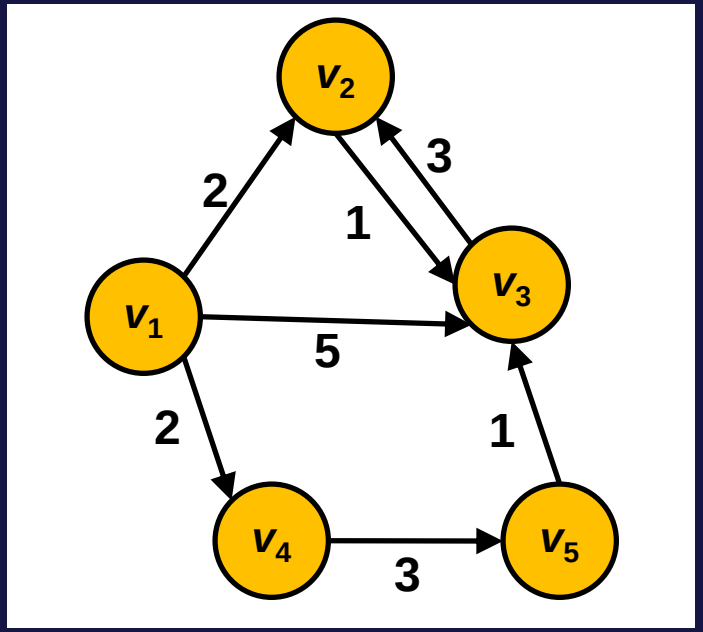
diagonalen = 0

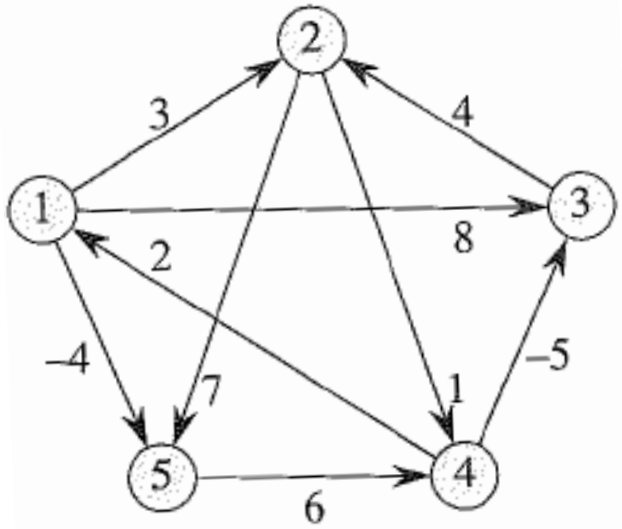
$L^{(m)}_{ij}$ = korteste afstand fra i til j for stier med m kanter
W = vægtmatricen

Tid $O(n^4)$

$L^{(3)}_{1,3}$?

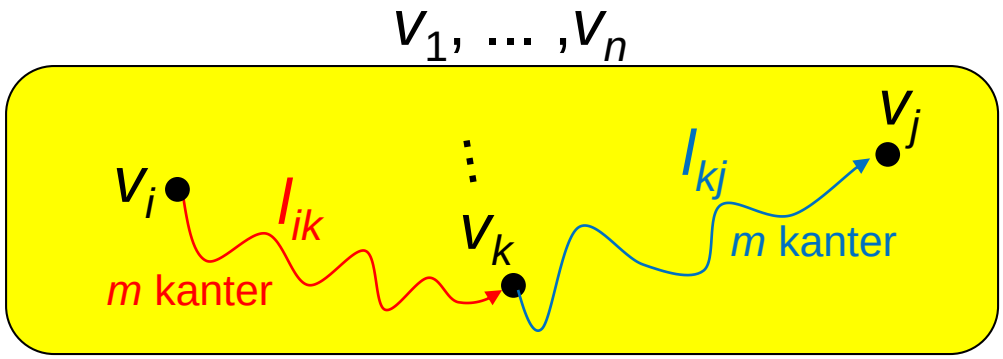
a) 0
b) 3
c) 5
d) 6
e) 9
f) $+\infty$
g) Ved ikke





$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$



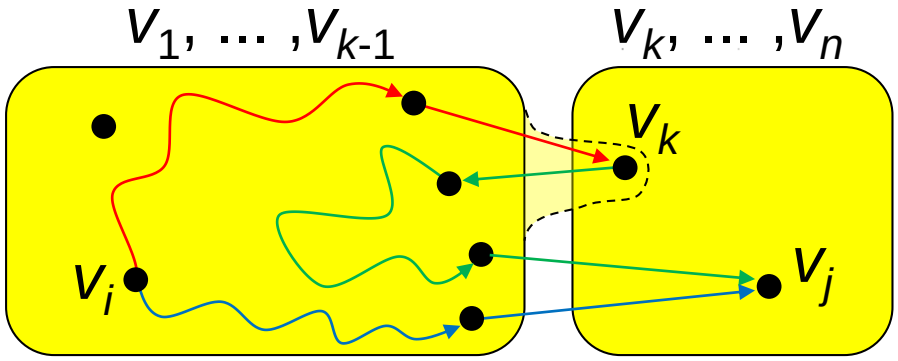
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

```
1  n = W.rows
2  L^(1) = W
3  m = 1
4  while m < n − 1
5      let L^(2m) be a new n × n matrix
6      L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7      m = 2m
8  return L^(m)
```

$L^{(m)}_{ij}$ = korteste afstand fra i til j for stier med m kanter
W = vægtmatricen

Tid $O(n^3 \cdot \log n)$

Floyd-Warshall



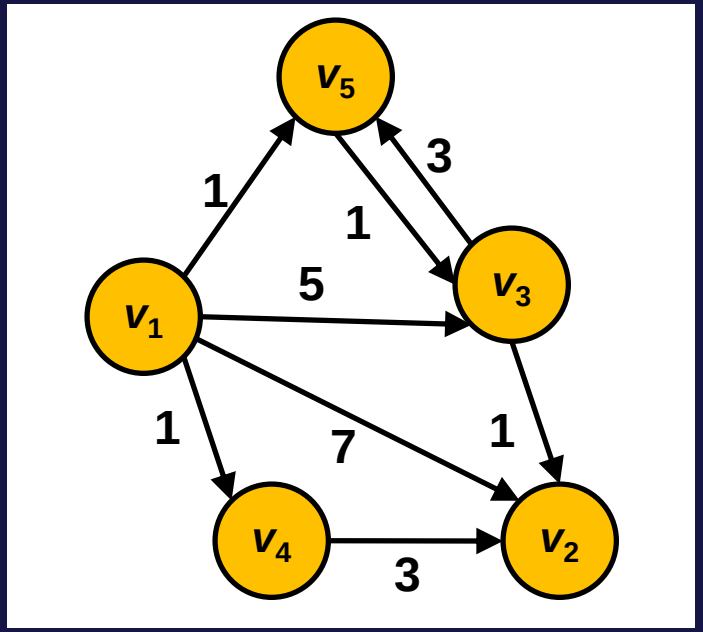

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

$d^{(k)}_{ij}$ = korteste vej fra *i* til *j* der kun går **via** 1..*k*

Tid $O(n^3)$

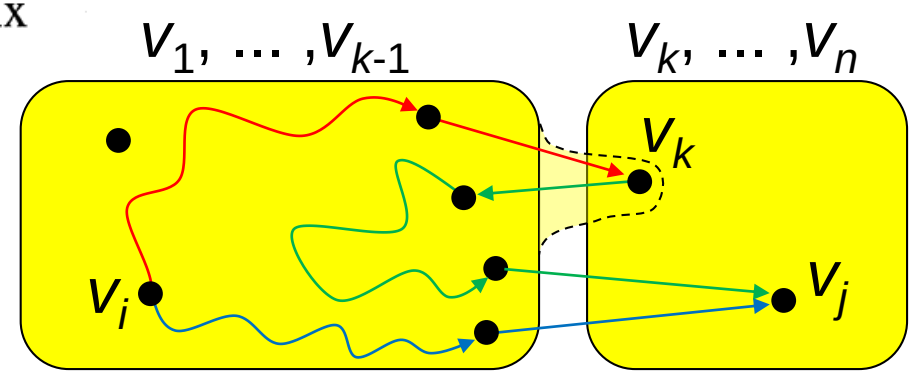
$d^{(3)}_{1,2}$?

a) 3
b) 4
c) 6
d) 7
e) +∞
f) Ved ikke



Transitive Lukning
(= Floyd-Warshall simplificeret)

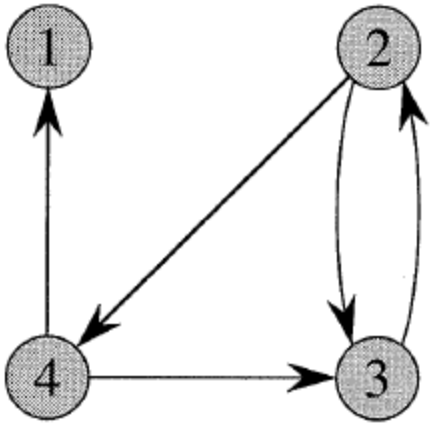
```
TRANSITIVE-CLOSURE(G)
 1  n = |G.V|
 2  let T^(0) = (t_ij^(0)) be a new n × n matrix
 3  for i = 1 to n
 4      for j = 1 to n
 5          if i == j or (i, j) ∈ G.E
 6              t_ij^(0) = 1
 7          else t_ij^(0) = 0
 8  for k = 1 to n
 9      let T^(k) = (t_ij^(k)) be a new n × n matrix
10      for i = 1 to n
11          for j = 1 to n
12              t_ij^(k) = t_ij^(k-1) ∨ (t_ik^(k-1) ∧ t_kj^(k-1))
13  return T^(n)
```



$t^{(k)}_{ij}$ = findes en vej fra i til j der kun går via $1..k$

Tid $O(n^3)$

Transitive Lukning: Eksempel



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Johnson's Algoritme

Korteste veje mellem alle par af knuder (APSP)

- Tynde grafer G, d.v.s. $m << n^2$,
- og postitive og negative vægtede kanter

Uden negative vægte ⇒ n x Dijkstra : $O(n \cdot (n \cdot \log n + m))$

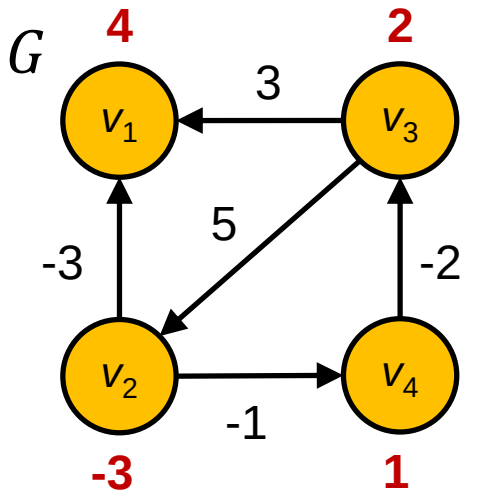
Med negative vægte ⇒ Floyd-Warshall : $O(n^3)$

Johnson's Algoritme

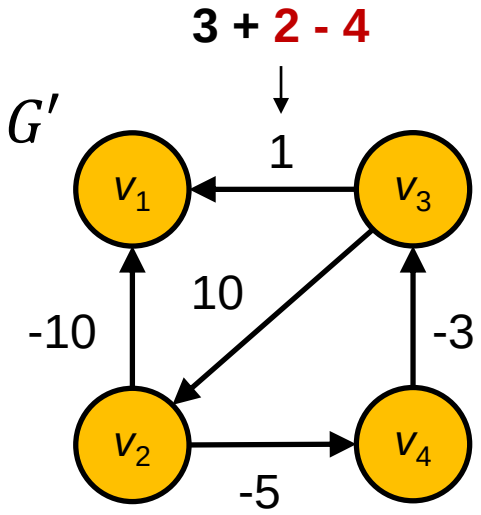
Ide: Kør Dijkstra på en graf G' uden negative kanter, som har de samme korteste veje som i G

Højde transformation

tilknyt hver knude u en arbitrær højde $h(u)$



$$w'(u,v) = w(u,v) + h(u) - h(v)$$



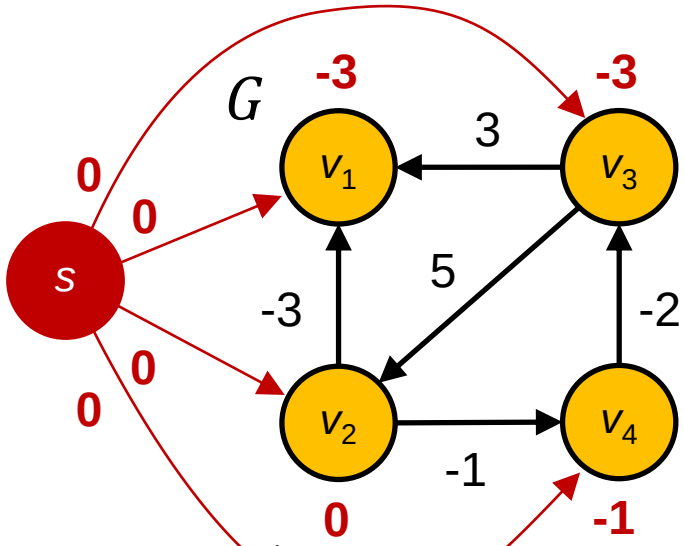
d	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-3	0	-3	-1
v_3	2	5	0	4
v_4	0	3	-2	0

$$l'(v_{i_1} v_{i_2} \cdots v_{i_k}) = l(v_{i_1} v_{i_2} \cdots v_{i_k}) + h(v_{i_1}) - h(v_{i_k})$$

Korteste vej i G $\Leftrightarrow$ korteste vej i G'

d'	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-10	0	-8	-5
v_3	0	10	0	5
v_4	-3	7	-3	0

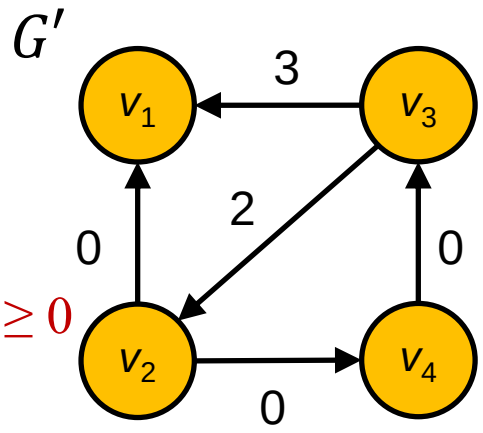
Johnson's Algoritme



$$w'(u, v) = w(u,v) + h(u) - h(v)$$

$$w'(u, v) = w(u, v) + h(u) - h(v) \geq 0$$

da $h(v) \leq h(u) + w(u, v)$



d	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-3	0	-3	-1
v_3	2	5	0	4
v_4	0	3	-2	0

d'	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	0	0	0	0
v_3	2	2	0	2
v_4	2	2	0	0

3 = 2 + 0 – (-1)

- Tilføj knude s og kanter fra s til alle knuder med vægt 0
- Lad $h(u) = d_G(s, u)$ – SSSP Bellman-Ford i tid $O(n \cdot m)$
- Kør Dijkstra fra hvert v_i i G' i tid $O(n \cdot (n \cdot \log n + m))$
- Lad $d_G(v_i, v_j) = d_{G'}(v_i, v_j) + h(v_j) - h(v_i)$

JOHNSON(G, w)

1 compute G', where $G'.V = G.V \cup \{s\}$,
  $G'.E = G.E \cup \{(s, v) : v \in G.V\}$, and
  $w(s, v) = 0$ for all $v \in G.V$
2 **if** BELLMAN-FORD(G', w, s) == FALSE
3 print "the input graph contains a negative-weight cycle"
4 **else for** each vertex $v \in G'.V$
5 set $h(v)$ to the value of $\delta(s, v)$
  computed by the Bellman-Ford algorithm
6 **for** each edge $(u, v) \in G'.E$
7 $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
8 let $D = (d_{uv})$ be a new $n \times n$ matrix
9 **for** each vertex $u \in G.V$
10 run DIJKSTRA$(G, \hat{w}, u)$ to compute $\hat{\delta}(u, v)$ for all $v \in G.V$
11 **for** each vertex $v \in G.V$
12 $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$
13 **return** D

Tid $O(n^2 \cdot \log n + n \cdot m)$

Korteste Veje

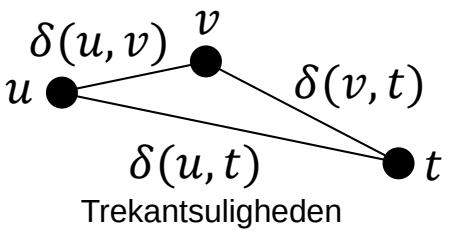
		En-til-alle korteste veje (SSSP)	Alle-til-alle korteste veje (APSP)
Acykliske grafer (positive og negative vægte)		$O(n+m)$	$O(n\cdot(n+m))$
Generelle grafer	Kun positive vægte	**Dijkstra** $O((n+m)\cdot\log n)$ (1) $O(n\cdot\log n + m)$ (2) $O(n^2)$ (3)	***n* x Dijkstra** $O(n^2\cdot\log n + n\cdot m)$ (2) $O(n^3)$ (3)
	Positive og negative vægte	**Bellman-Ford** $O(n\cdot m)$	**Floyd-Warshall** $O(n^3)$ **Johnson** $O(n^2\cdot\log n + n\cdot m)$ (2)

Forskellige prioritetskøer: (1) binær heap, (2) Fibonacci heap, (3) array

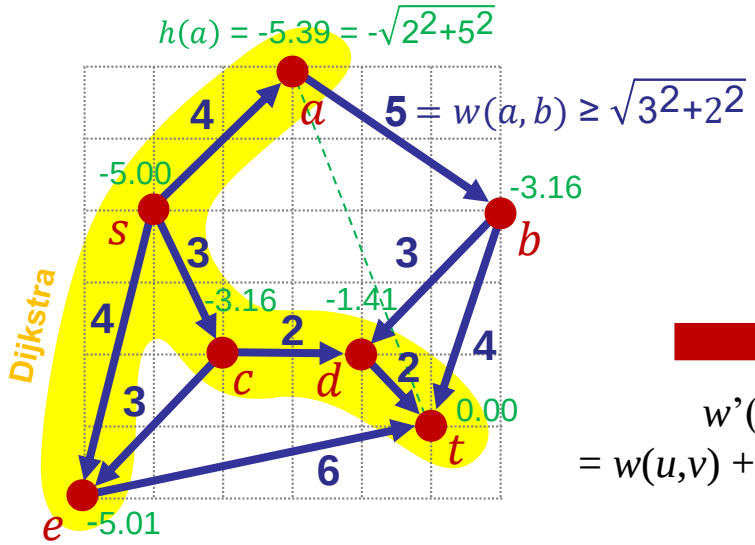
IKKE PENSUM

A*-algoritmen : korteste vej $s \rightsquigarrow t$

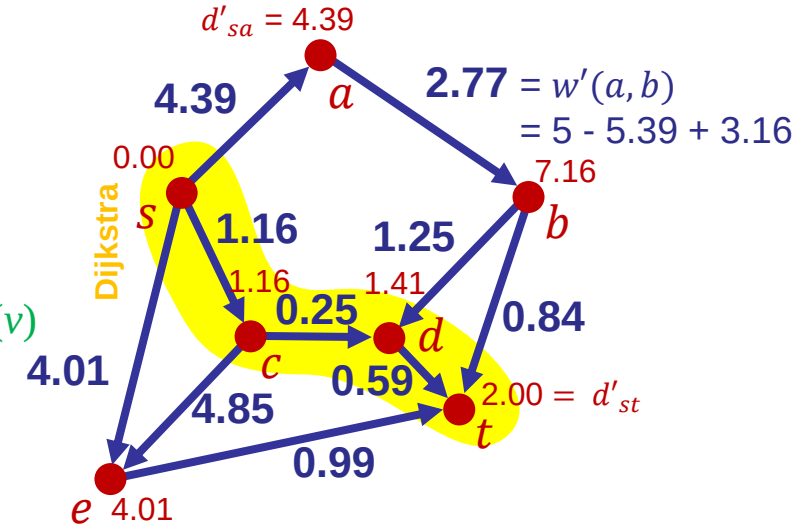
- Antag knude u har position (u_x, u_y) og $w(u,v) \geq \delta(u,v) = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2}$
- $h(u) = -\delta(u,t) \Rightarrow w'(u,v) = w(u,v) + h(u) - h(v) \geq \delta(u,v) - \delta(u,t) + \delta(v,t) \geq 0$
- $d_{st} = d'_{st} - h(s) + h(t) = 2.00 + 5.00 - 0.00 = \underline{\mathbf{7.00}}$



Trekantsuligheden



$w'(u,v) = w(u,v) + h(u) - h(v)$



A* reducerer i praksis antallet af knuder Dijkstra's algoritme besøger

Youtube: Compare A* with Dijkstra algorithm