Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Graphalgorithmen

Bisherige Ergebnisse

- Dijkstras Algorithmus für positive Kantengewichte
- Bellman-Ford für allgemeine Kantengewichte; Laufzeit $\mathbf{O}(|V|^2 + |V| \cdot |E|)$
- Negative Zyklen können erkannt werden

Heute

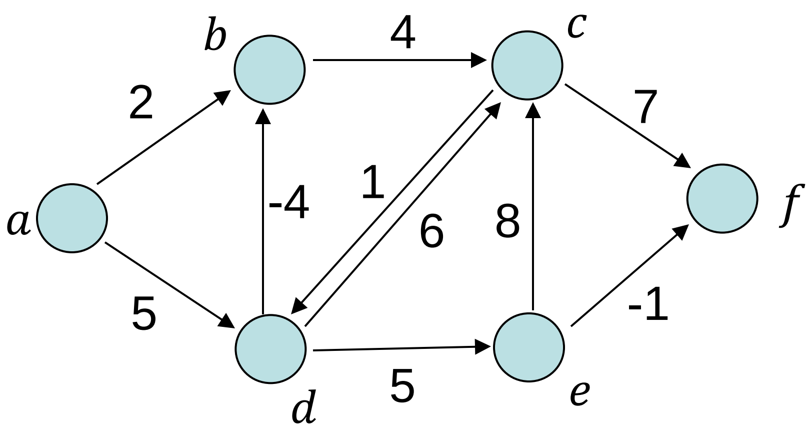
All Pairs Shortest Paths (mit negativen Kantengewichten)

Graphalgorithmen

All Pairs Shortest Path (APSP)

- Eingabe: Gewichteter Graph $G = (V, E)$
- Ausgabe: Für jedes Paar von Knoten $u, v \in V$ die Distanz von u nach v sowie einen kürzesten Weg

	a	b	c	d	e	f
a	0	1	5	5	10	9
b	∞	0	4	5	10	9
c	∞	-3	0	1	6	5
d	∞	-4	0	0	5	4
e	∞	5	8	9	0	-1
f	∞	∞	∞	∞	∞	0

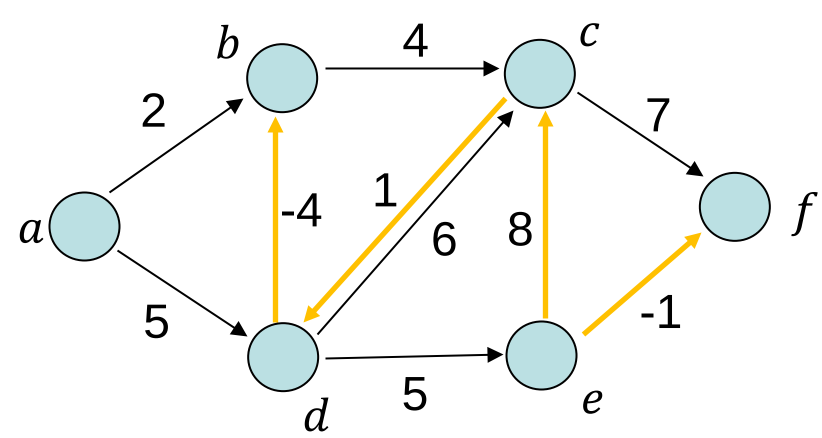


Graphalgorithmen

All Pairs Shortest Path (APSP)

- Eingabe: Gewichteter Graph $G = (V, E)$
- Ausgabe: Für jedes Paar von Knoten $u, v \in V$ die Distanz von u nach v sowie einen kürzesten Weg

	a	b	c	d	e	f
a	0	1	5	5	10	9
b	∞	0	4	5	10	9
c	∞	-3	0	1	6	5
d	∞	-4	0	0	5	4
e	∞	5	8	9	0	-1
f	∞	∞	∞	∞	∞	0



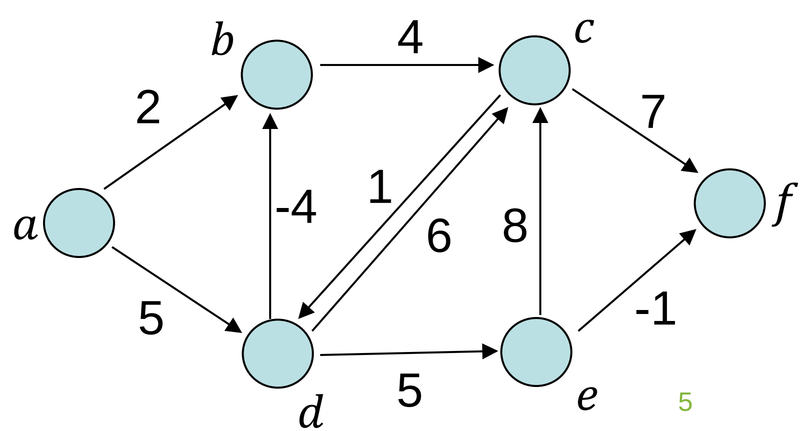
Graphalgorithmen

Eingabe APSP

Matrix $W = (w_{ij})$,die Graph repräsentiert

$$w_{ij} = \begin{cases} 0 & \text{, wenn } i = j \\ \text{Gewicht der ger. Kante } (i,j) & \text{, wenn } i \neq j \text{ und } (i,j) \in E \\ \infty & \text{, wenn } i \neq j \text{ und } (i,j) \notin E \end{cases}$$

	a	b	c	d	e	f
a	0	2	∞	5	∞	∞
b	∞	0	4	∞	∞	∞
c	∞	∞	0	1	∞	7
d	∞	-4	6	0	5	∞
e	∞	∞	8	∞	0	-1
f	∞	∞	∞	∞	∞	0



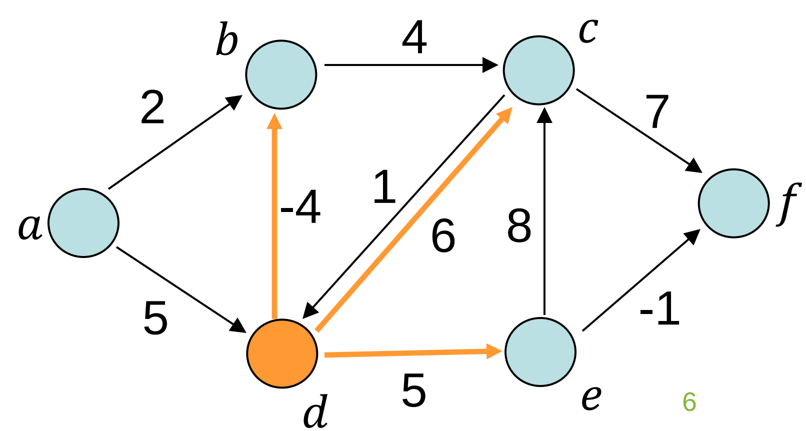
Graphalgorithmen

Eingabe APSP

Matrix $W = (w_{ij})$,die Graph repräsentiert

$$w_{ij} = \begin{cases} 0 & \text{, wenn } i = j \\ \text{Gewicht der ger. Kante } (i,j) & \text{, wenn } i \neq j \text{ und } (i,j) \in E \\ \infty & \text{, wenn } i \neq j \text{ und } (i,j) \notin E \end{cases}$$

	a	b	c	d	e	f
a	0	2	∞	5	∞	∞
b	∞	0	4	∞	∞	∞
c	∞	∞	0	1	∞	7
d	∞	-4	6	0	5	∞
e	∞	∞	8	∞	0	-1
f	∞	∞	∞	∞	∞	0



Graphalgorithmen

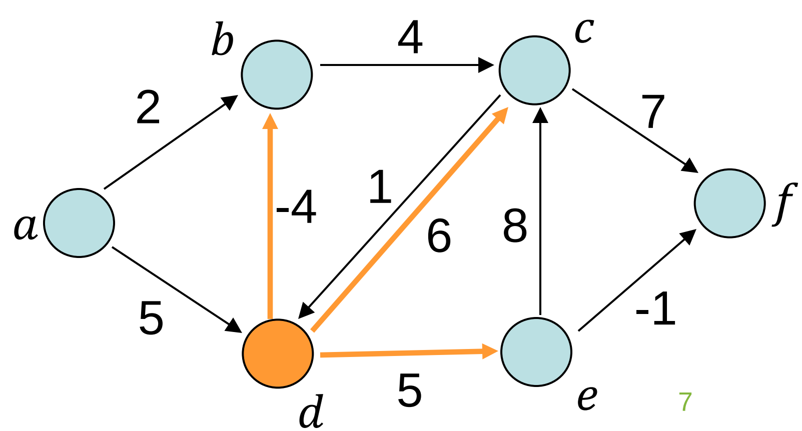
Annahme:
Keine negativen Zyklen!

Eingabe APSP

Matrix $W = (w_{ij})$,die Graph repräsentiert

$$w_{ij} = \begin{cases} 0 & \text{, wenn } i = j \\ \text{Gewicht der ger. Kante } (i,j) & \text{, wenn } i \neq j \text{ und } (i,j) \in E \\ \infty & \text{, wenn } i \neq j \text{ und } (i,j) \notin E \end{cases}$$

	a	b	c	d	e	f
a	0	2	∞	5	∞	∞
b	∞	0	4	∞	∞	∞
c	∞	∞	0	1	∞	7
d	∞	-4	6	0	5	∞
e	∞	∞	8	∞	0	-1
f	∞	∞	∞	∞	∞	0



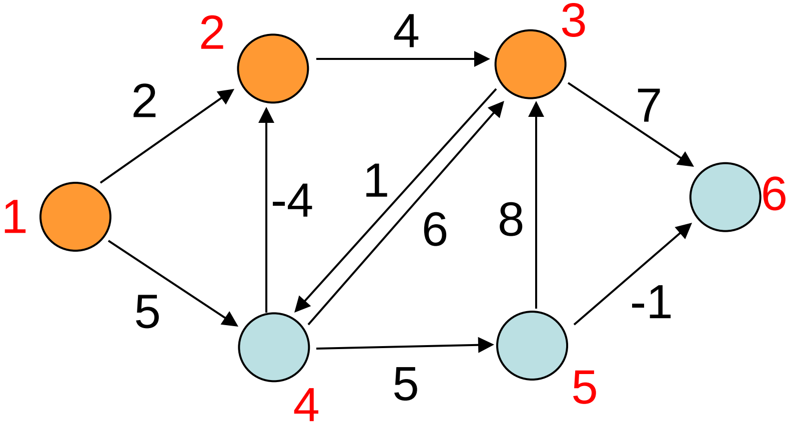
Graphalgorithmen

Eine neue Rekursion:

- Nummeriere Knoten von 1 bis $n = |V|$
- Betrachte kürzeste i-j-Wege, die nur über Knoten 1 bis k laufen

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	∞	-4	0	0	5	7
5	∞	∞	8	14	0	-1
6	∞	∞	∞	∞	∞	0

$k = 3$



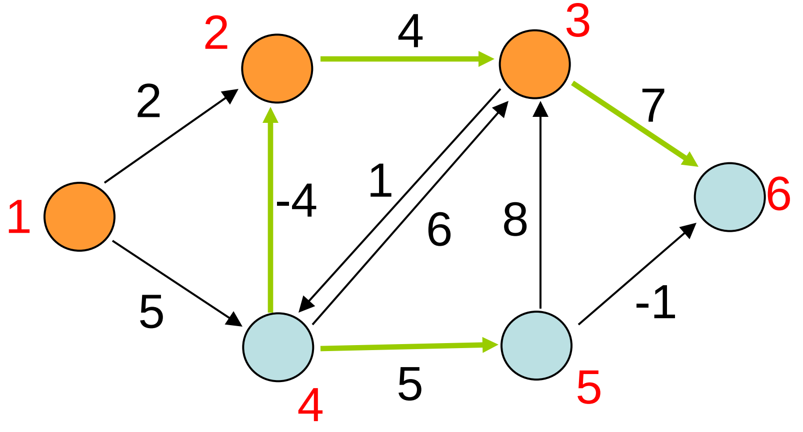
Graphalgorithmen

Eine neue Rekursion:

- Nummeriere Knoten von 1 bis $n = |V|$
- Betrachte kürzeste i-j-Wege, die nur über Knoten 1 bis k laufen

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	∞	-4	0	0	5	7
5	∞	∞	8	14	0	-1
6	∞	∞	∞	∞	∞	0

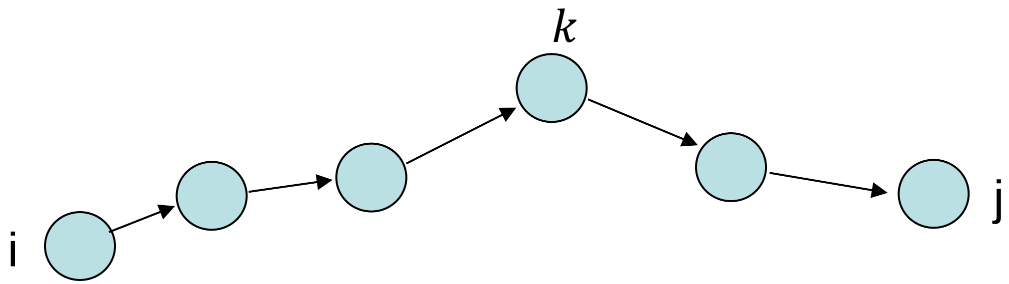
$k = 3$



Graphalgorithmen

Zur Erinnerung

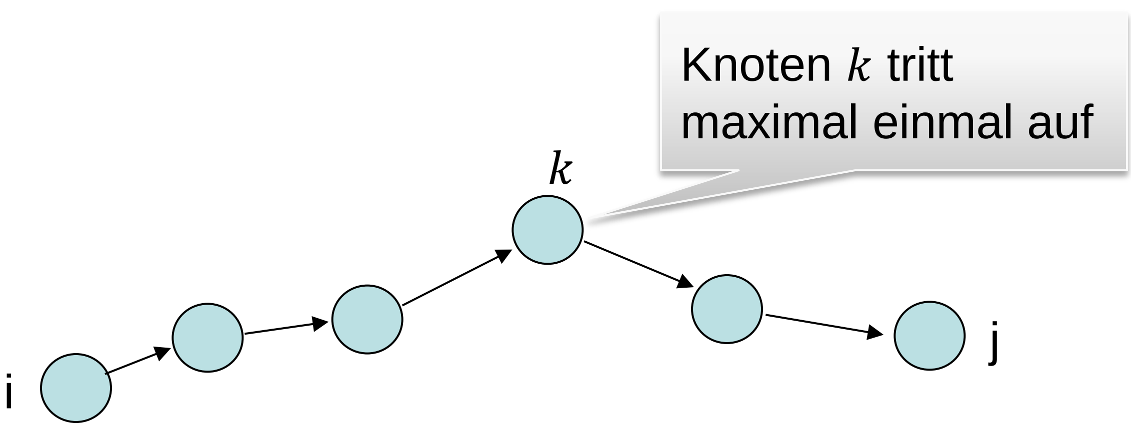
- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 53)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus $\{1, \dots, k\}$ läuft:



Graphalgorithmen

Zur Erinnerung

- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 53)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus $\{1, \ldots, k\}$ läuft:

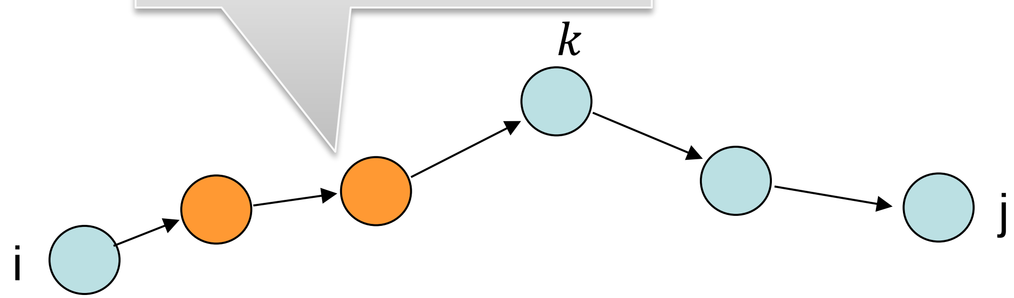


Graphalgorithmen

Zur Erinnerung

- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 53)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte noten aus $\{1, \dots, k\}$ läuft:

Weg von u nach k führt nur über Knoten aus $\{1, \dots, k-1\}$

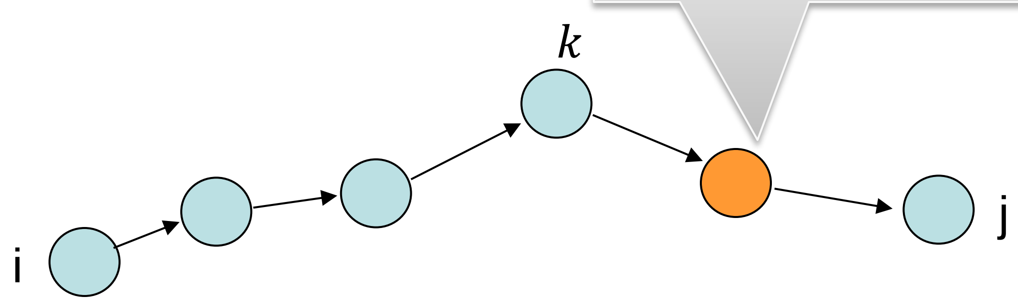


Graphalgorithmen

Zur Erinnerung

- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 53)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über ft:

Weg von k nach v führt nur über Knoten aus $\{1, \dots, k-1\}$

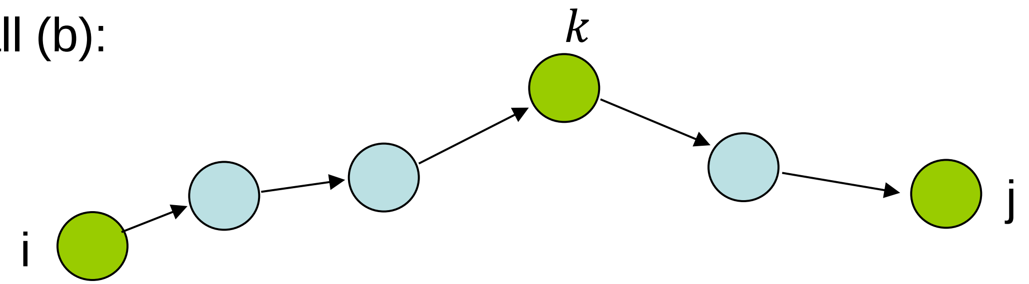


Graphalgorithmen

Die Rekursion

- Kürzester i-j-Weg über Knoten aus $\{1, \ldots, k\}$ ist
- (a) kürzester i-j-Weg über Knoten aus $\{1, \ldots, k-1\}$ oder
- (b) kürzester i-k-Weg über Knoten aus $\{1, \ldots, k-1\}$ gefolgt von kürzestem k-j-Weg über Knoten aus $\{1, \ldots, k-1\}$

Fall (b):



Graphalgorithmen

Die Rekursion

- Sei $d_{ij}^{(k)}$ die Länge eines kürzesten i-j-Wegs über Knoten aus $\{1, \dots, k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{, falls } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{, falls } k \geq 1 \end{cases}$$

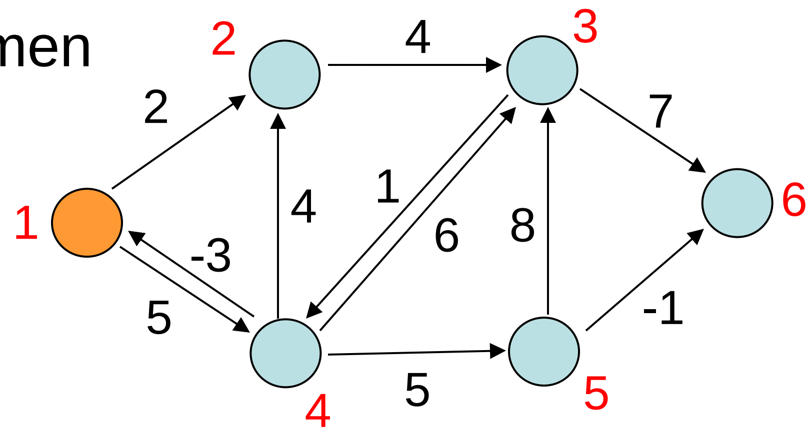
- Matrix $\mathrm{D}^{(\mathrm{n})} = (d_{ij}^{(n)})$ enthält die gesuchte Lösung

Graphalgorithmen

Floyd-Warshall(W, n)

1. $\mathrm{D}^{(0)} \leftarrow \mathrm{W}$
2. **for** $k \leftarrow 1$ **to** n **do**
3. **for** $i \leftarrow 1$ **to** n **do**
4. **for** $j \leftarrow 1$ **to** n **do**
5. $\mathrm{d}_{\mathrm{ij}}^{(\mathrm{k})} \leftarrow \min\left(\mathrm{d}_{\mathrm{ij}}^{(\mathrm{k}-1)}, \mathrm{d}_{\mathrm{ik}}^{(\mathrm{k}-1)} + \mathrm{d}_{\mathrm{kj}}^{(\mathrm{k}-1)}\right)$
6. **return** $\mathrm{D}^{(\mathrm{n})}$

Graphalgorithmen



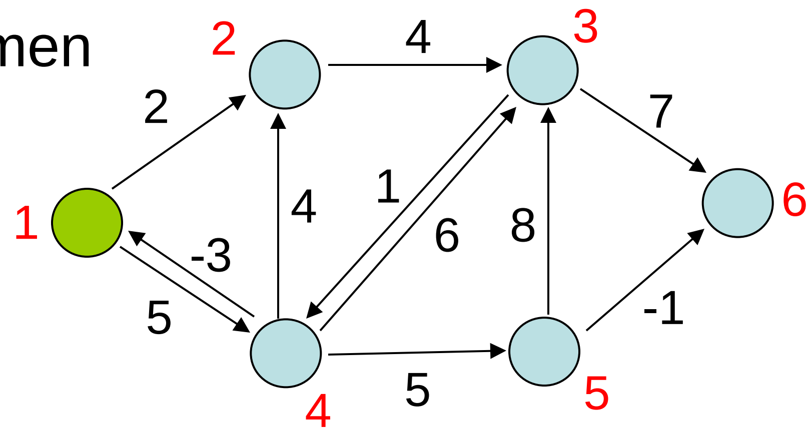
$D^{(0)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	4	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(1)}$

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

Graphalgorithmen



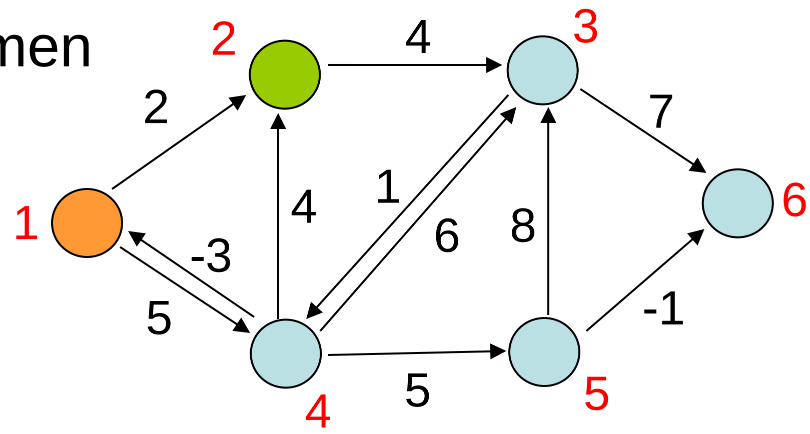
$D^{(0)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	4	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(1)}$

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

Graphalgorithmen



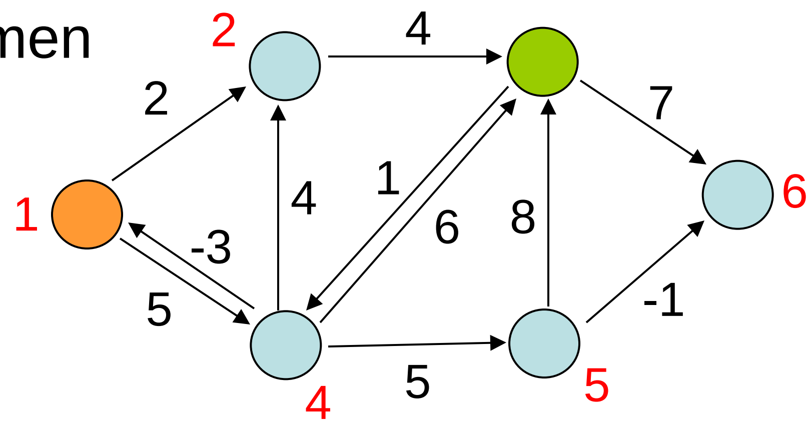
	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	4	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(0)}$

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

$D^{(1)}$

Graphalgorithmen



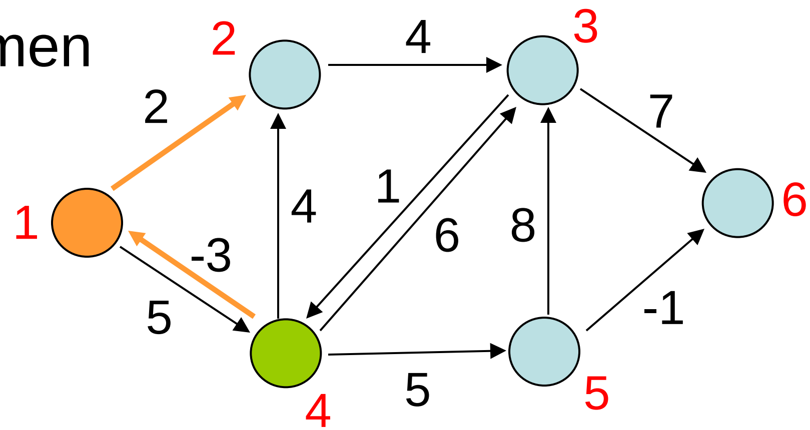
	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	4	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(0)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4						
5						
6						

$D^{(1)}$

Graphalgorithmen



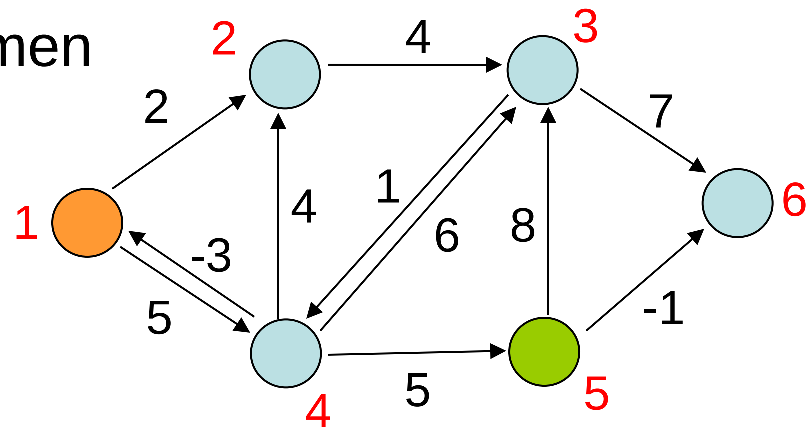
$D^{(0)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	4	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(1)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5						
6						

Graphalgorithmen



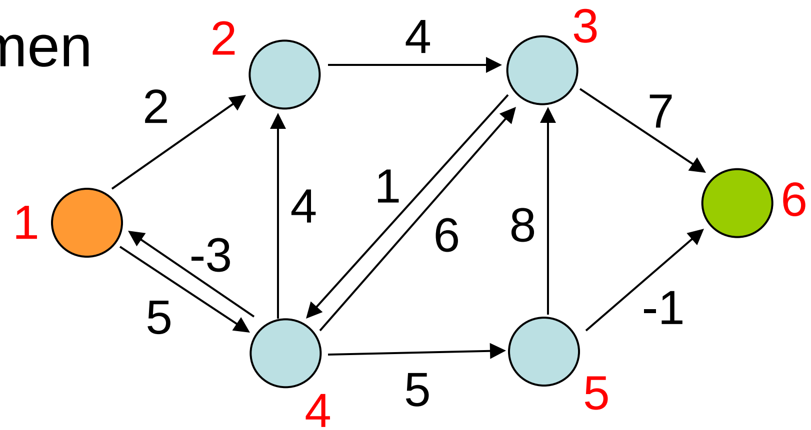
	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	4	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(0)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6						

$D^{(1)}$

Graphalgorithmen



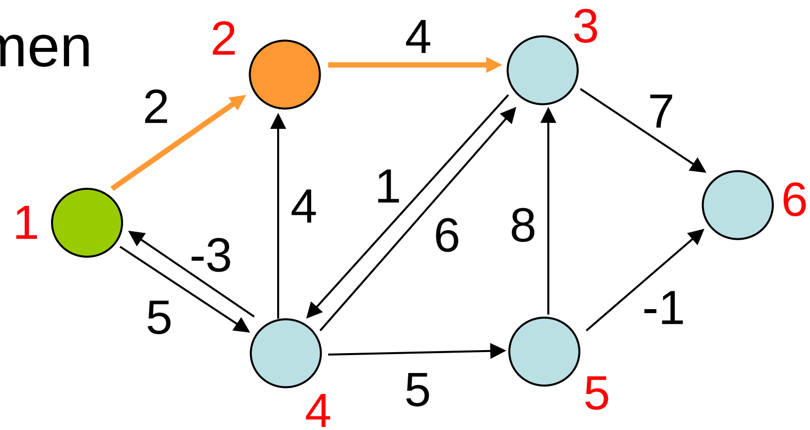
$D^{(0)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	4	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(1)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

Graphalgorithmen



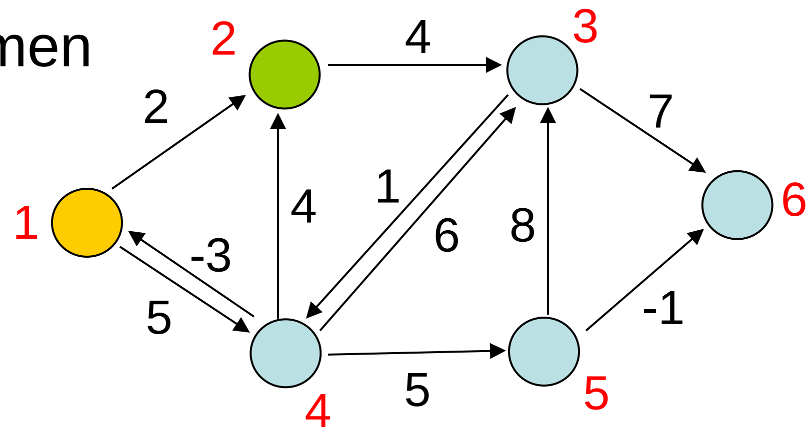
$D^{(1)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(2)}$

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

Graphalgorithmen



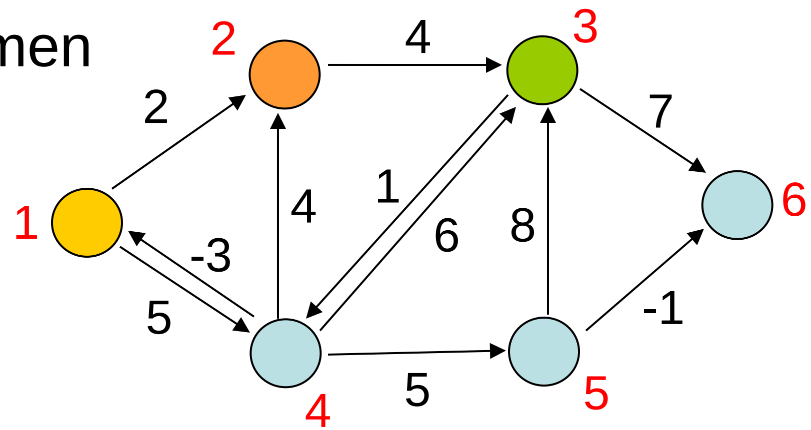
$D^{(1)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(2)}$

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

Graphalgorithmen



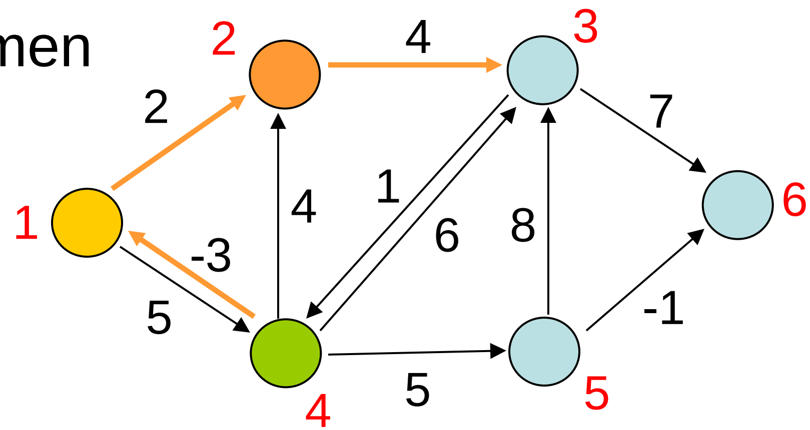
$D^{(1)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4						
5						
6						

Graphalgorithmen



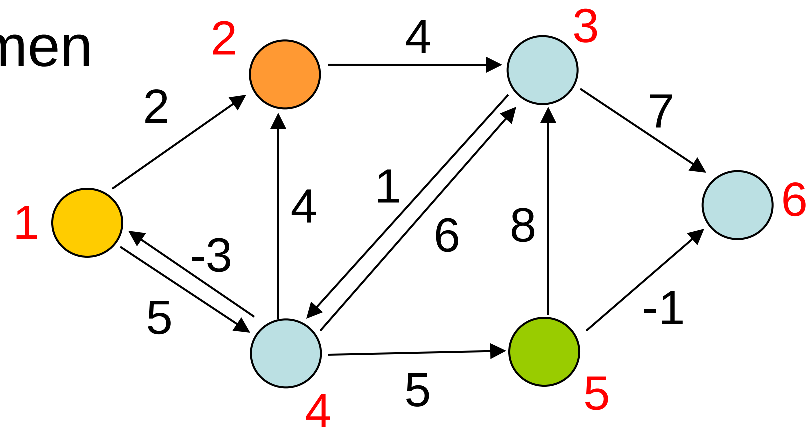
$D^{(1)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5						
6						

Graphalgorithmen



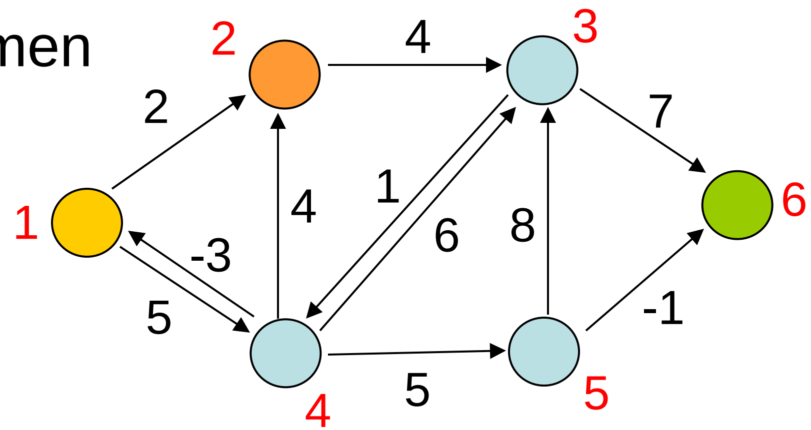
$D^{(1)}$

	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6						

Graphalgorithmen



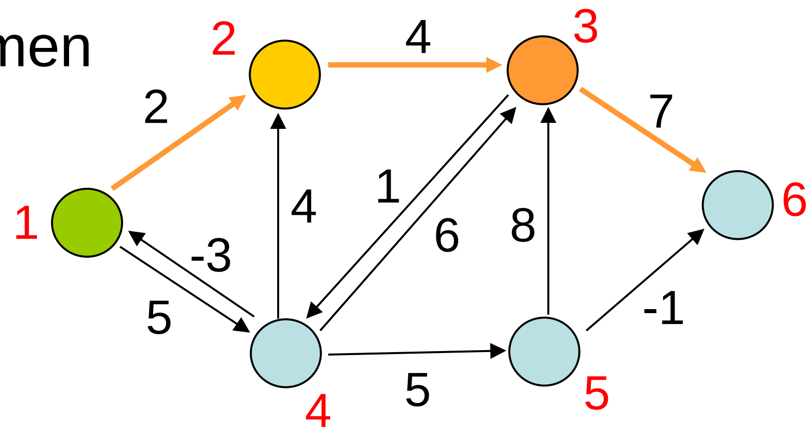
	1	2	3	4	5	6
1	0	2	∞	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	6	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(1)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(2)}$

Graphalgorithmen



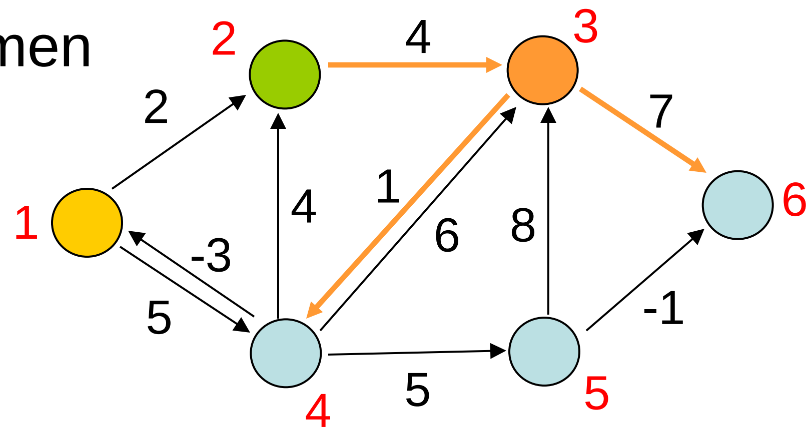
$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2						
3						
4						
5						
6						

Graphalgorithmen



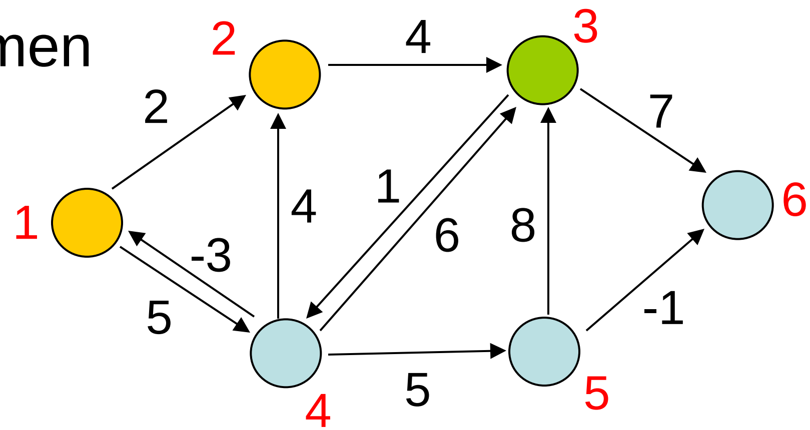
$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3						
4						
5						
6						

Graphalgorithmen



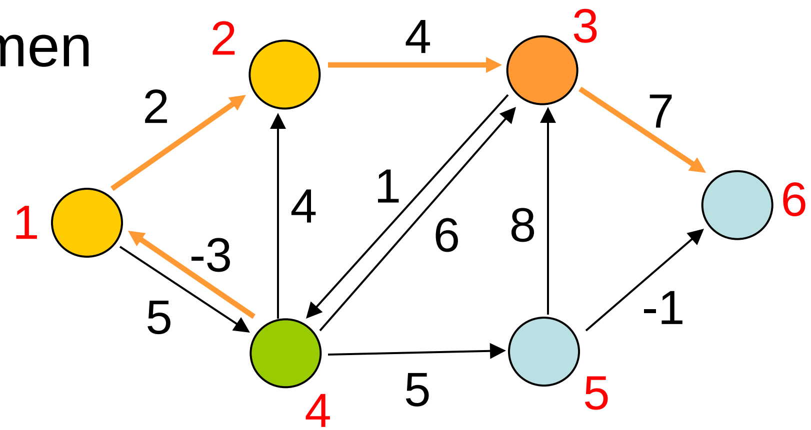
$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4						
5						
6						

Graphalgorithmen



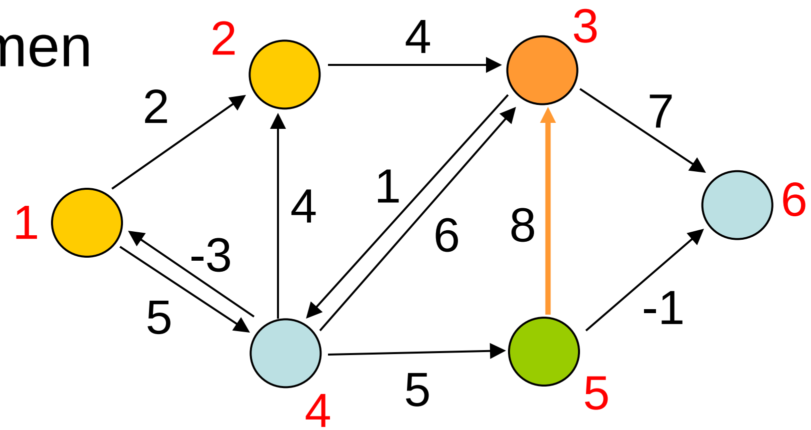
$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5						
6						

Graphalgorithmen



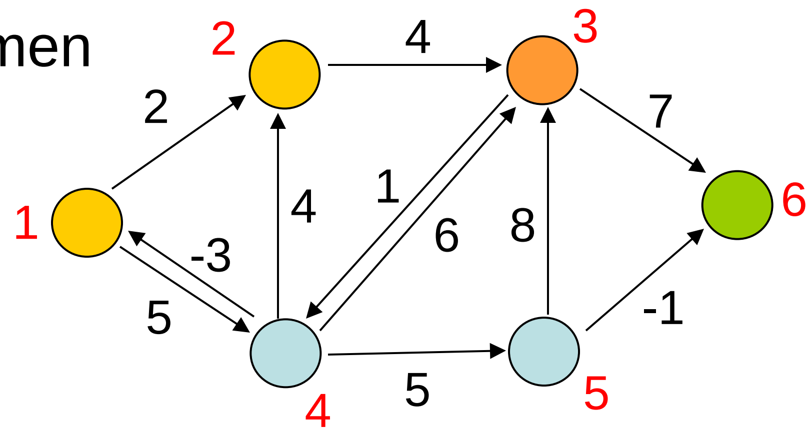
$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6						

Graphalgorithmen



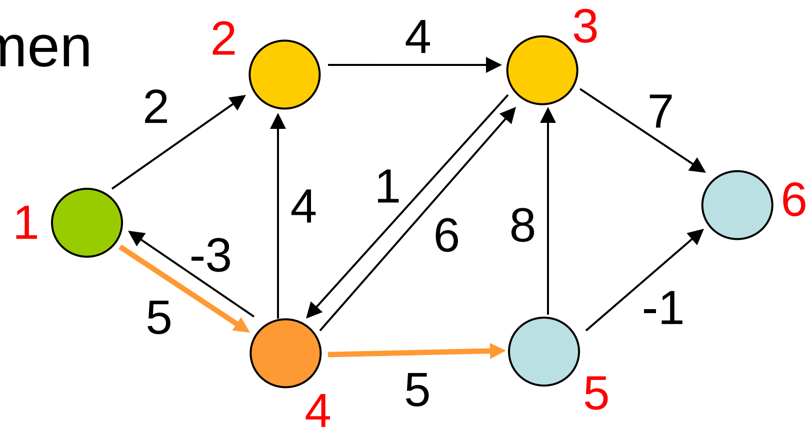
	1	2	3	4	5	6
1	0	2	6	5	∞	∞
2	∞	0	4	∞	∞	∞
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	∞
5	∞	∞	8	∞	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(2)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(3)}$

Graphalgorithmen



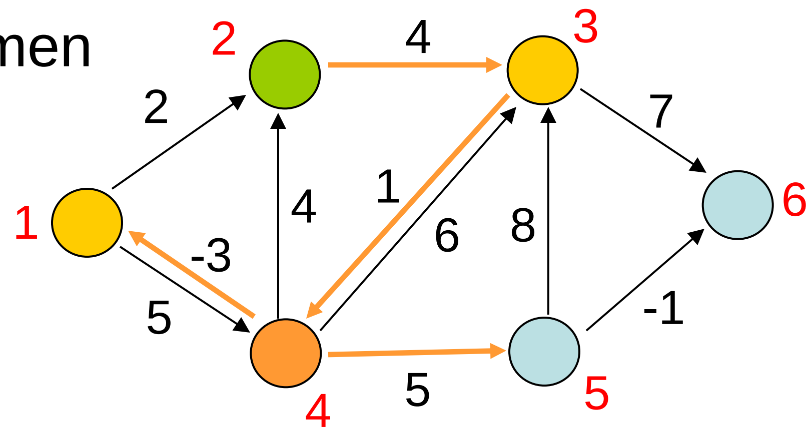
$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2						
3						
4						
5						
6						

Graphalgorithmen



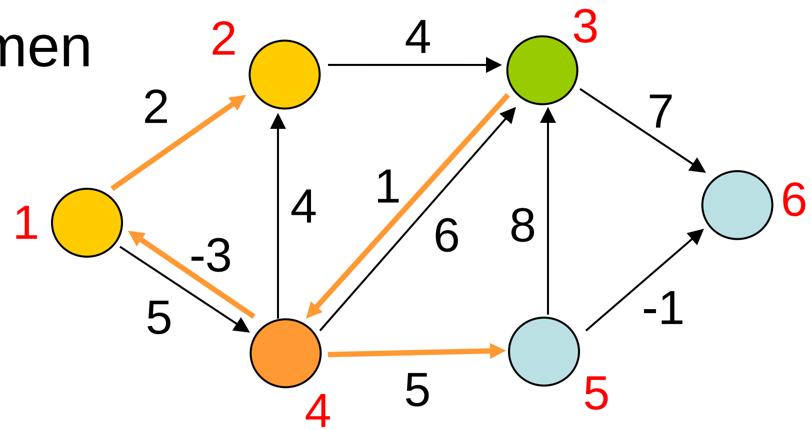
$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3						
4						
5						
6						

Graphalgorithmen



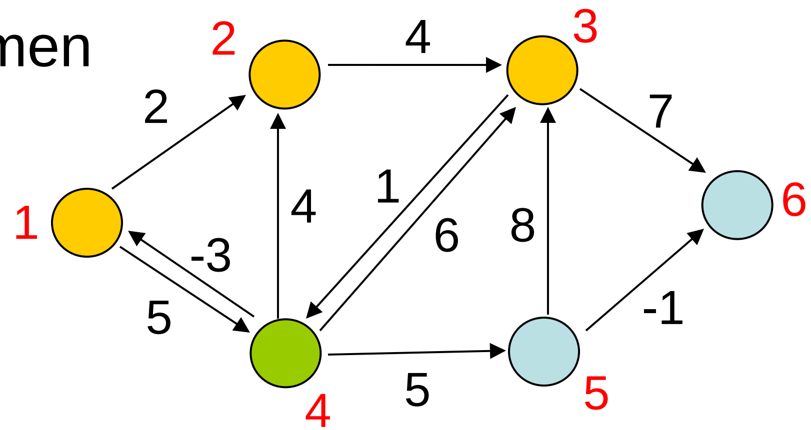
$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4						
5						
6						

Graphalgorithmen



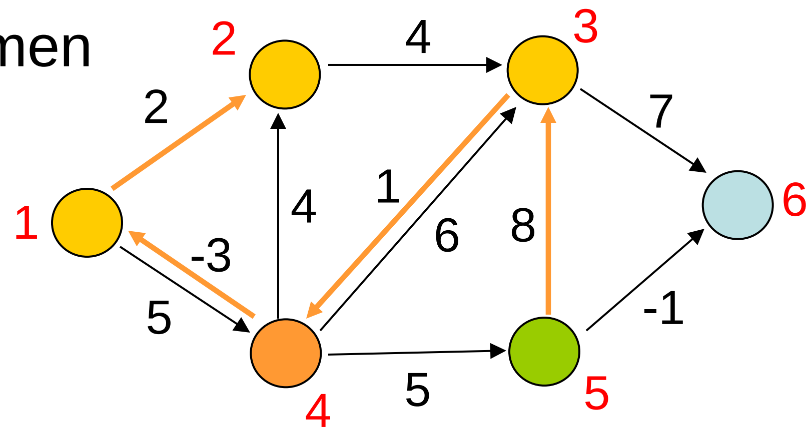
$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5						
6						

Graphalgorithmen



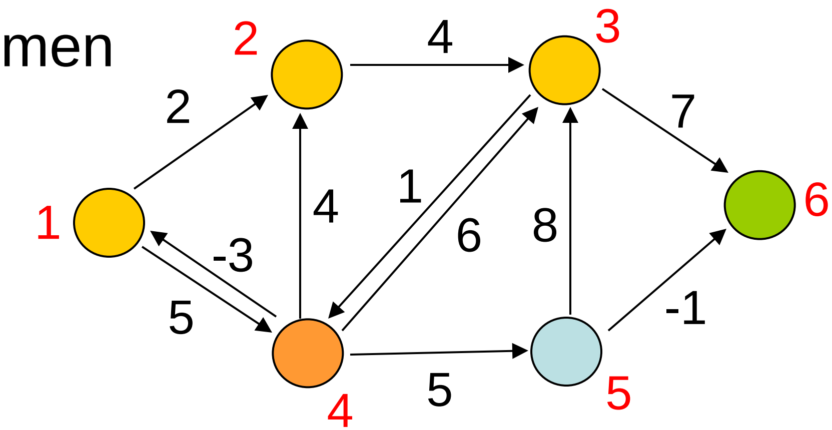
$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6						

Graphalgorithmen



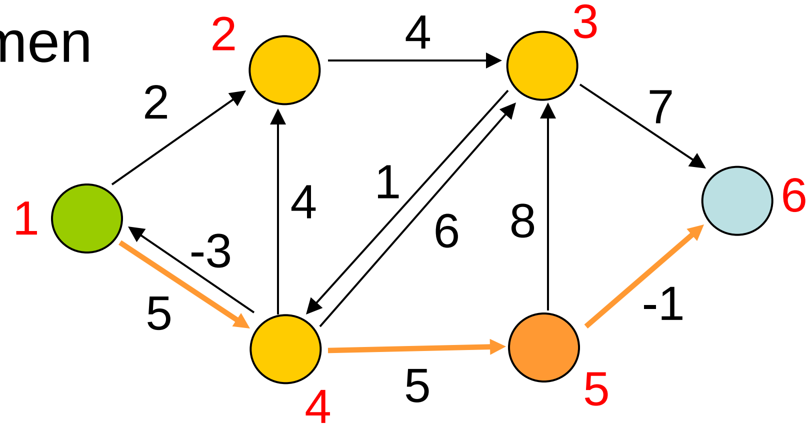
$D^{(3)}$

	1	2	3	4	5	6
1	0	2	6	5	∞	13
2	∞	0	4	5	∞	11
3	∞	∞	0	1	∞	7
4	-3	-1	3	0	5	10
5	∞	∞	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

Graphalgorithmen



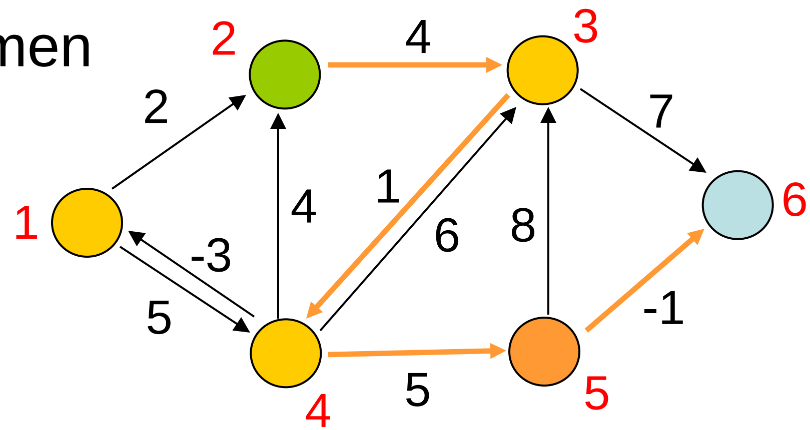
$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2						
3						
4						
5						
6						

Graphalgorithmen



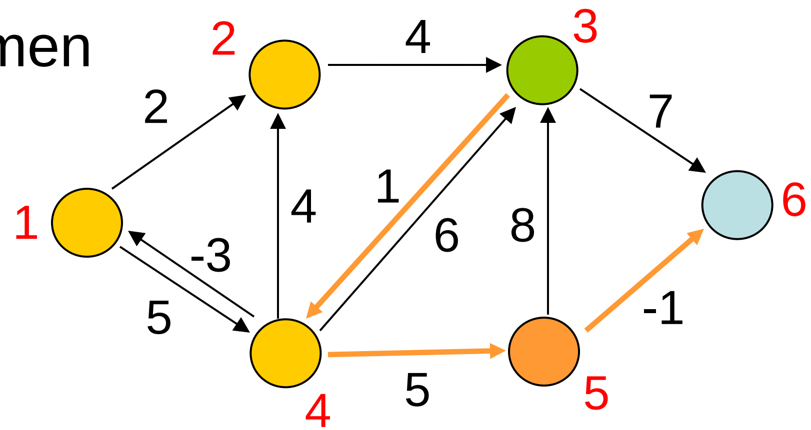
$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3						
4						
5						
6						

Graphalgorithmen



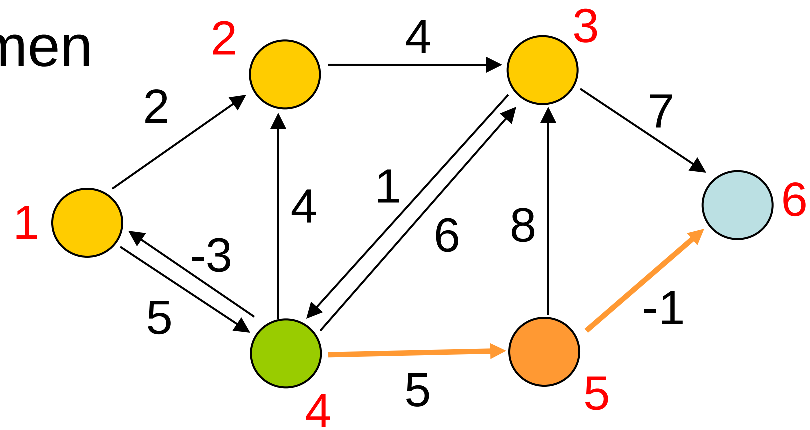
$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4						
5						
6						

Graphalgorithmen



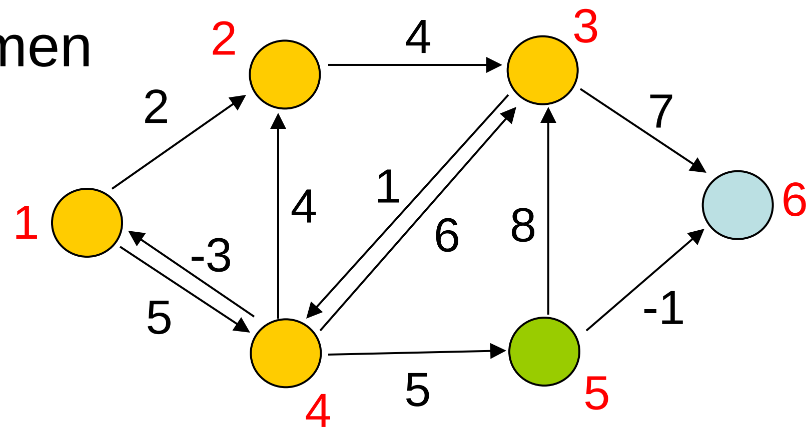
$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4	-3	-1	3	0	5	4
5						
6						

Graphalgorithmen



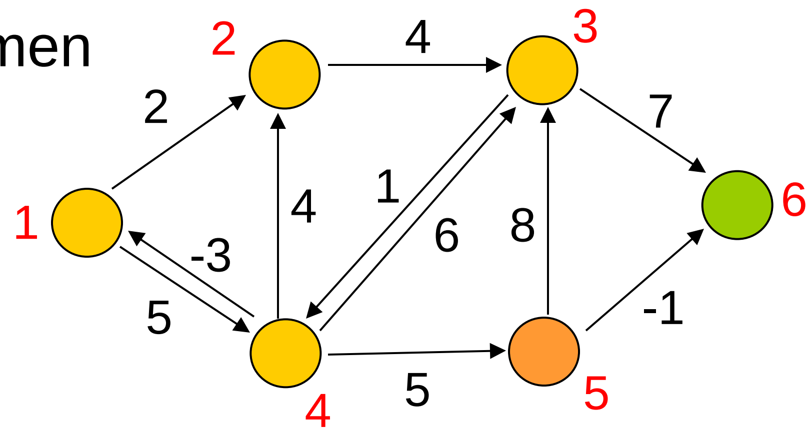
$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4	-3	-1	3	0	5	4
5	6	8	8	9	0	-1
6						

Graphalgorithmen



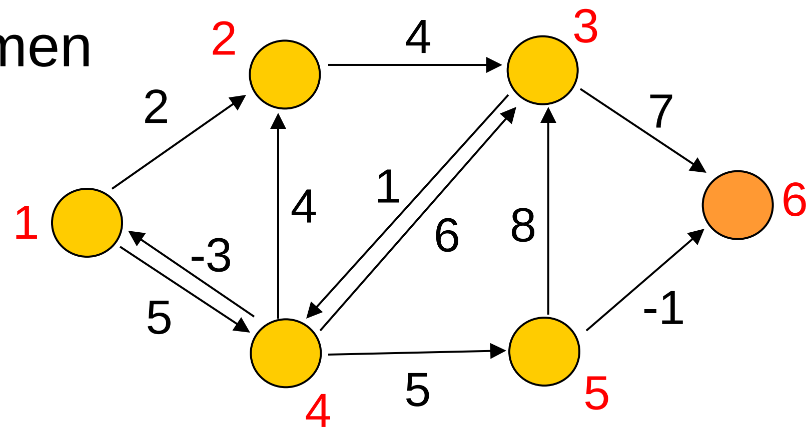
$D^{(4)}$

	1	2	3	4	5	6
1	0	2	6	5	10	13
2	2	0	4	5	10	11
3	-2	0	0	1	6	7
4	-3	-1	3	0	5	10
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4	-3	-1	3	0	5	4
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

Graphalgorithmen



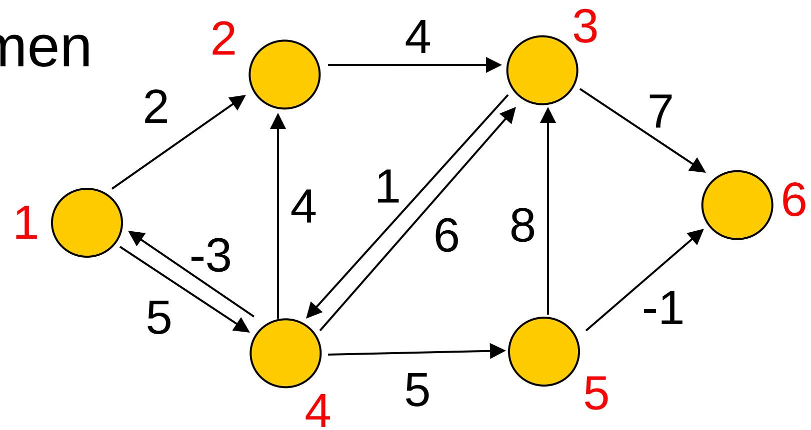
$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4	-3	-1	3	0	5	4
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(6)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4	-3	-1	3	0	5	4
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

Graphalgorithmen



$D^{(5)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4	-3	-1	3	0	5	4
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

$D^{(6)}$

	1	2	3	4	5	6
1	0	2	6	5	10	9
2	2	0	4	5	10	9
3	-2	0	0	1	6	5
4	-3	-1	3	0	5	4
5	6	8	8	9	0	-1
6	∞	∞	∞	∞	∞	0

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- Die Laufzeit folgt sofort, da 3 ineinander geschachtelte Schleifen jeweils von 1 bis $|V| = n$ durchlaufen werden.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- Die Laufzeit folgt sofort, da 3 ineinander geschachtelte Schleifen jeweils von 1 bis $|V| = n$ durchlaufen werden.
- Korrektheit per Induktion über k. Z.z. die Matrix $D^{(k)}$ enthält die kürzeste Entfernung zwischen allen Paaren von Knoten, wenn die Pfade nur über die Knoten $\{1, .., k\}$ verlaufen dürfen.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- Die Laufzeit folgt sofort, da 3 ineinander geschachtelte Schleifen jeweils von 1 bis $|V| = n$ durchlaufen werden.
- Korrektheit per Induktion über k. Z.z. die Matrix $D^{(k)}$ enthält die kürzeste Entfernung zwischen allen Paaren von Knoten, wenn die Pfade nur über die Knoten $\{1, .., k\}$ verlaufen dürfen.
- (I.A.) Die Matrix $D^{(0)}$ ist gleich der Eingabematrix W und enthält somit alle Wege im Graphen, die keinen Zwischenknoten benutzen.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- Die Laufzeit folgt sofort, da 3 ineinander geschachtelte Schleifen jeweils von 1 bis $|V| = n$ durchlaufen werden.
- Korrektheit per Induktion über k. Z.z. die Matrix $D^{(k)}$ enthält die kürzeste Entfernung zwischen allen Paaren von Knoten, wenn die Pfade nur über die Knoten $\{1, .., k\}$ verlaufen dürfen.
- (I.A.) Die Matrix $D^{(0)}$ ist gleich der Eingabematrix W und enthält somit alle Wege im Graphen, die keinen Zwischenknoten benutzen.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- (I.S.) Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten $\{1, \ldots, k\}$ verlaufen.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- (I.S.) Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten $\{1, \dots, k\}$ verlaufen. Da G keine negativen Zyklen hat, gibt es immer einen kürzesten Weg, der jeden Knoten nur maximal einmal besucht.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- (I.S.) Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten $\{1, \dots, k\}$ verlaufen. Da G keine negativen Zyklen hat, gibt es immer einen kürzesten Weg, der jeden Knoten nur maximal einmal besucht. Verläuft der kürzeste Weg von i nach j nun über Knoten k, so ist seine Länge nach (I.V.) genau $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- (I.S.) Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten $\{1, \ldots, k\}$ verlaufen. Da G keine negativen Zyklen hat, gibt es immer einen kürzesten Weg, der jeden Knoten nur maximal einmal besucht. Verläuft der kürzeste Weg von i nach j nun über Knoten k, so ist seine Länge nach (I.V.) genau $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Ansonsten verläuft er nicht über k und somit nur über die Knoten $\{1, \ldots, k-1\}$.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- (I.S.) Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten $\{1, \dots, k\}$ verlaufen. Da G keine negativen Zyklen hat, gibt es immer einen kürzesten Weg, der jeden Knoten nur maximal einmal besucht. Verläuft der kürzeste Weg von i nach j nun über Knoten k, so ist seine Länge nach (I.V.) genau $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Ansonsten verläuft er nicht über k und somit nur über die Knoten $\{1, \dots, k-1\}$. Damit ist seine Länge $d_{ij}^{(k-1)}$.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- (I.S.) Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten $\{1, \dots, k\}$ verlaufen. Da G keine negativen Zyklen hat, gibt es immer einen kürzesten Weg, der jeden Knoten nur maximal einmal besucht. Verläuft der kürzeste Weg von i nach j nun über Knoten k, so ist seine Länge nach (I.V.) genau $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Ansonsten verläuft er nicht über k und somit nur über die Knoten $\{1, \dots, k-1\}$. Damit ist seine Länge $d_{ij}^{(k-1)}$. Somit wird die Länge durch die Rekursion korrekt berechnet.

Graphalgorithmen

Satz 58

Sei $G = (V, E)$ ein Graph ohne negativen Zyklen. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $\mathbf{O}(|V|^3)$ Zeit.

Beweis

- (I.V.) Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- (I.S.) Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten $\{1, \dots, k\}$ verlaufen. Da G keine negativen Zyklen hat, gibt es immer einen kürzesten Weg, der jeden Knoten nur maximal einmal besucht. Verläuft der kürzeste Weg von i nach j nun über Knoten k, so ist seine Länge nach (I.V.) genau $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Ansonsten verläuft er nicht über k und somit nur über die Knoten $\{1, \dots, k-1\}$. Damit ist seine Länge $d_{ij}^{(k-1)}$. Somit wird die Länge durch die Rekursion korrekt berechnet.

Graphalgorithmen

Aufrechterhalten der kürzesten Wege:

- Konstruiere Vorgängermatrix Π
- Dazu konstruiere Sequenz $\Pi^{(1)}, \dots, \Pi^{(n)}$ mit $\Pi = \Pi^{(n)}$
- $\Pi^{(k)}$ ist Vorgängermatrix zu $D^{(k)}$
- $\pi_{ij}^{(k)}$ ist Vorgänger von Knoten j auf dem kürzesten Weg von Knoten i über Knoten aus $\{1, \dots, k\}$
- Die Startmatrix:

$$\pi_{ij}^{(0)} = \begin{cases} \mathbf{nil} & \text{, falls } i = j \text{ oder } w_{ij} = \infty \\ i & \text{, falls } i \neq j \text{ und } w_{ij} < \infty \end{cases}$$

Graphalgorithmen

Aufrechterhalten der kürzesten Wege:

- Konstruiere Vorgängermatrix Π
- Dazu konstruiere Sequenz $\Pi^{(1)}, \dots, \Pi^{(n)}$ mit $\Pi = \Pi^{(n)}$
- $\Pi^{(k)}$ ist Vorgängermatrix zu $D^{(k)}$
- $\pi_{ij}^{(k)}$ ist Vorgänger von Knoten j auf dem kürzesten Weg von Knoten i über Knoten aus $\{1, \dots, k\}$
- Das Aktualisieren:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{, falls } d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{, falls } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Graphalgorithmen

SSSP (pos. Kantengewichte)

Dijkstra; Laufzeit $\mathbf{O}\big((|V| + |E|)\log|V|\big)$

SSSP (allgemeine Kantengewichte)

Bellman-Ford; Laufzeit $\mathbf{O}(|V|^2 + |V| \cdot |E|)$

APSP (allgemeine Kantengewichte, keine negativen Zyklen)

Floyd-Warshall; Laufzeit $\mathbf{O}(|V|^3)$

Graphalgorithmen

Binäre Relationen

- Eine (binäre) Relation R zwischen Elementen der Mengen A und B ist eine Teilmenge von $A \times B$.
- Ist $A = B$, so spricht man auch von einer Relation auf der Menge A

Graphen und Relationen

- Sei $G = (V, E)$ ein gerichteter Graph (möglicherweise mit Selbstschleifen, d.h. Kanten von v nach v)
- E ist Teilmenge von $V \times V$
- Also kann man E als Relation auf der Menge V interpretieren

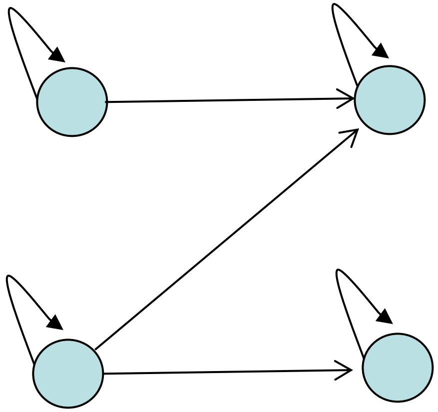
Graphalgorithmen

Reflexivität

Eine Relation R auf der Menge V heißt reflexiv, wenn $(v, v) \in R$ für alle $v \in V$.

Interpretation als Graph

- Alle Knoten haben Selbstschleifen



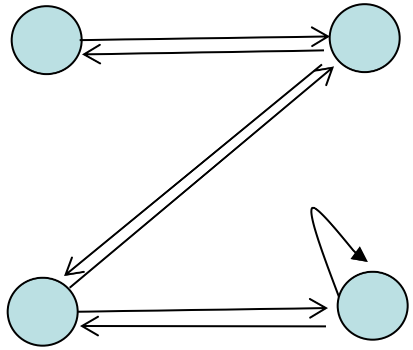
Graphalgorithmen

Symmetrie

Eine Relation R auf der Menge V heißt symmetrisch, wenn aus $(u, v) \in R$ folgt, dass $(v, u) \in R$ ist.

Interpretation als Graph

- Alle Kanten sind in beiden Richtungen vorhanden (ungerichteter Graph)
- Selbstschleifen erlaubt



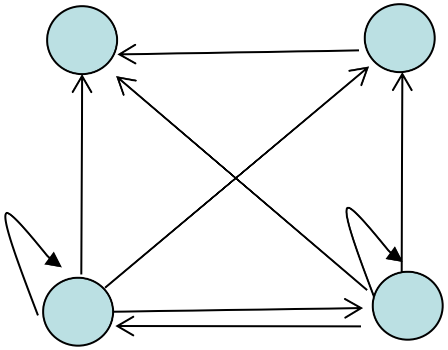
Graphalgorithmen

Transitivität

Eine Relation R auf der Menge V heißt transitiv, wenn aus $(u, v) \in R$ und $(v, w) \in R$ folgt, dass $(u, w) \in R$ ist.

Interpretation als Graph

- Man kann zwei aufeinanderfolgende Kanten immer abkürzen
- Wiederholtes Anwenden:
  Gibt es Weg von u nach v,
  so gibt es direkte Verbindung



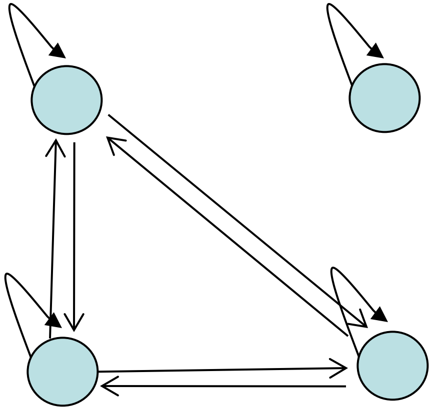
Graphalgorithmen

Äquivalenzrelation

Eine Relation R auf der Menge V heißt Äquivalenzrelation, wenn sie reflexiv, symmetrisch und transitiv ist.

Interpretation als Graph

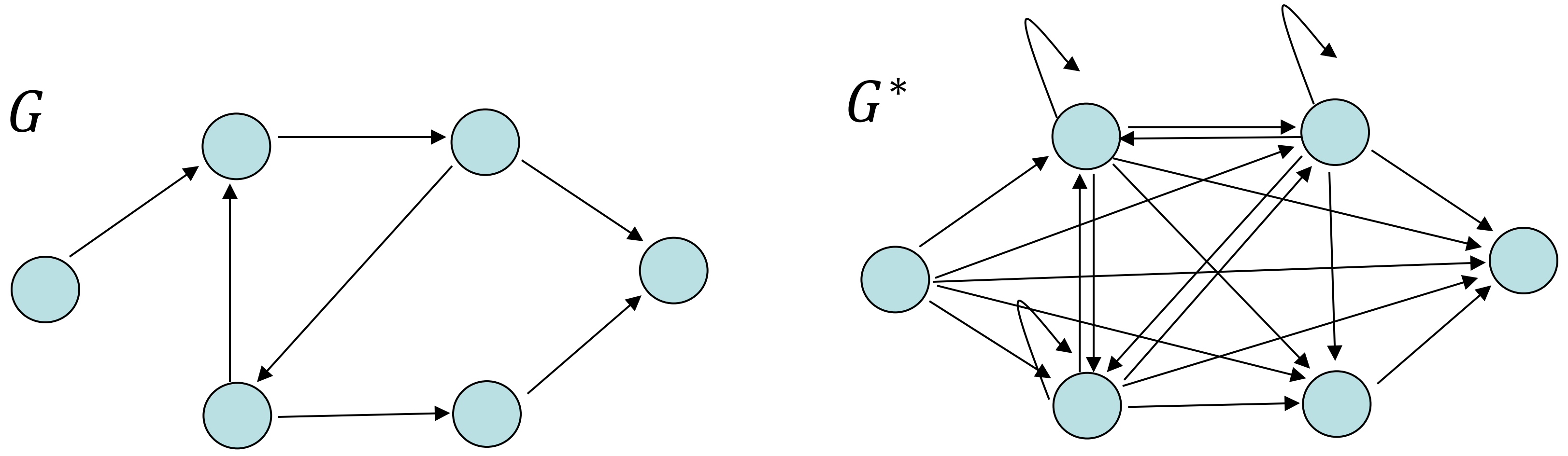
- Ungerichteter Graph mit Selbstschleifen an jedem Knoten
- Zusammenhangskomponenten sind vollständig verbunden



Graphalgorithmen

Das transitive Hülle Problem

- Gegeben sei ein gerichteter, ungewichteter Graph $G = (V, E)$
- Gesucht: Die transitive Hülle $G^* = (V, E^*)$ von G, wobei
  $E^* = \{(u, v)\colon \text{ es gibt Weg von } u \text{ nach } v \text{ in } G\}$



Graphalgorithmen

Transitive Hülle

- in $\mathbf{O}(|V|^3)$ Zeit mit Floyd-Warshall
- In $\mathbf{O}(|V|^2 + |V|\,|E|)$ Zeit mit Breiten- oder Tiefensuche von jedem Knoten
- Geht das auch schneller?

Graphalgorithmen

Graphen und Matrixmultiplikation

- Sei A die $n \times n$-Adjazenzmatrix eines ungerichteten Graphen G mit Knotenmenge $\{1, \ldots, n\}$
- Was ist $A \cdot A$?

Graphalgorithmen

Behauptung 59

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Graphalgorithmen

Behauptung 59

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Beweis

„⇒“ Es gilt bei der Matrixmultiplikation, dass $z_{ij} = \sum a_{ik} \cdot a_{kj}$.

Graphalgorithmen

Behauptung 59

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Beweis

„⇒“ Es gilt bei der Matrixmultiplikation, dass $z_{ij} = \sum a_{ik} \cdot a_{kj}$. Da die Einträge von A entweder 0 oder 1 sind, folgt aus $z_{ij} > 0$,
dass für mindestens einen Index k gilt: $a_{ik} = 1$ und $a_{kj} = 1$.

Graphalgorithmen

Behauptung 59

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Beweis

„⇒“ Es gilt bei der Matrixmultiplikation, dass $z_{ij} = \sum a_{ik} \cdot a_{kj}$. Da die Einträge von A entweder 0 oder 1 sind, folgt aus $z_{ij} > 0$,
dass für mindestens einen Index k gilt: $a_{ik} = 1$ und $a_{kj} = 1$. Damit gibt es aber eine Kante von i nach k und k nach j, also einen Pfad der Länge 2.

Graphalgorithmen

Behauptung 59

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Beweis

„⇒“ Es gilt bei der Matrixmultiplikation, dass $z_{ij} = \sum a_{ik} \cdot a_{kj}$. Da die Einträge von A entweder 0 oder 1 sind, folgt aus $z_{ij} > 0$,
dass für mindestens einen Index k gilt: $a_{ik} = 1$ und $a_{kj} = 1$. Damit gibt es aber eine Kante von i nach k und k nach j, also einen Pfad der Länge 2.

„⇐“ Gibt es einen Pfad der Länge 2 von i nach j, so verläuft dieser über einen weiteren Knoten, sagen wir k. Damit ist $a_{ik} = 1$ und $a_{kj} = 1$.

Graphalgorithmen

Behauptung 59

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Beweis

„⇒“ Es gilt bei der Matrixmultiplikation, dass $z_{ij} = \sum a_{ik} \cdot a_{kj}$. Da die Einträge von A entweder 0 oder 1 sind, folgt aus $z_{ij} > 0$,
dass für mindestens einen Index k gilt: $a_{ik} = 1$ und $a_{kj} = 1$. Damit gibt es aber eine Kante von i nach k und k nach j, also einen Pfad der Länge 2.

„⇐“ Gibt es einen Pfad der Länge 2 von i nach j, so verläuft dieser über einen weiteren Knoten, sagen wir k. Damit ist $a_{ik} = 1$ und $a_{kj} = 1$. Da die Einträge von A entweder 0 oder 1 sind, folgt $z_{ij} = \sum a_{ik} \cdot a_{kj}$.

Graphalgorithmen

Behauptung 59

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Beweis

„⇒" Es gilt bei der Matrixmultiplikation, dass $z_{ij} = \sum a_{ik} \cdot a_{kj}$. Da die Einträge von A entweder 0 oder 1 sind, folgt aus $z_{ij} > 0$,
dass für mindestens einen Index k gilt: $a_{ik} = 1$ und $a_{kj} = 1$. Damit gibt es aber eine Kante von i nach k und k nach j, also einen Pfad der Länge 2.

„⇐" Gibt es einen Pfad der Länge 2 von i nach j, so verläuft dieser über einen weiteren Knoten, sagen wir k. Damit ist $a_{ik} = 1$ und $a_{kj} = 1$. Da die Einträge von A entweder 0 oder 1 sind, folgt $z_{ij} = \sum a_{ik} \cdot a_{kj}$.

Graphalgorithmen

Behauptung 60

Sei $Z' = A \cdot A + A$. Dann gilt, dass $z'_{ij} > 0$, g.d.w. es einen Weg der Länge 1 oder 2 von Knoten i zu Knoten j gibt.

Beweis

Folgt sofort aus dem vorhergehenden Lemma und der Tatsache, dass A genau für die Paare i, j einen Eintrag 1 hat, die durch eine Kante (einen Weg der Länge 1) verbunden sind

Konstruiere Matrix B mit:

$$b_{ij} = 1 \Leftrightarrow z'_{ij} > 0$$

Graphalgorithmen

Behauptung 61

Der durch Matrix B definierte Graph hat einen Weg von Knoten i nach j, g.d.w. der durch Matrix A definierte Graph einen Weg der Länge höchstens 2 zwischen i und j hat.

Beweis

Folgt aus der vorherigen Behauptung.

Graphalgorithmen

Behauptung 62

Sei P ein Weg der Länge $k > 1$ in G (dem Graph mit Adjazenzmatrix A) von Knoten i zu Knoten j. Dann gibt es in dem von Matrix B beschriebenen Graphen G‘ einen Weg von i nach j mit Länge maximal $\frac{2}{3}k$.

Graphalgorithmen

Behauptung 62

Sei P ein Weg der Länge $k > 1$ in G (dem Graph mit Adjazenzmatrix A) von Knoten i zu Knoten j. Dann gibt es in dem von Matrix B beschriebenen Graphen G' einen Weg von i nach j mit Länge maximal $\frac{2}{3}k$.

Beweis

- Sei P ein Weg in G mit k Kanten.

Graphalgorithmen

Behauptung 62

Sei P ein Weg der Länge $k > 1$ in G (dem Graph mit Adjazenzmatrix A) von Knoten i zu Knoten j. Dann gibt es in dem von Matrix B beschriebenen Graphen G‘ einen Weg von i nach j mit Länge maximal $\frac{2}{3}k$.

Beweis

- Sei P ein Weg in G mit k Kanten.
- Ist k gerade, dann gibt es in G‘ einen Weg der Länge $k/2$, da man jeweils zwei aufeinanderfolgende Kanten von P durch eine Kante in G‘ ersetzen kann.

Graphalgorithmen

Behauptung 62

Sei P ein Weg der Länge $k > 1$ in G (dem Graph mit Adjazenzmatrix A) von Knoten i zu Knoten j. Dann gibt es in dem von Matrix B beschriebenen Graphen G‘ einen Weg von i nach j mit Länge maximal $\frac{2}{3}k$.

Beweis

- Sei P ein Weg in G mit k Kanten.
- Ist k gerade, dann gibt es in G‘ einen Weg der Länge $k/2$, da man jeweils zwei aufeinanderfolgende Kanten von P durch eine Kante in G‘ ersetzen kann.
- Ist k ungerade, dann gibt es in G‘ einen Weg der Länge $\lceil k/2 \rceil$, da man bis auf die letzte Kante jeweils zwei aufeinanderfolgende Kanten von P durch eine Kante in G‘ ersetzen kann.

Graphalgorithmen

Behauptung 62

Sei P ein Weg der Länge $k > 1$ in G (dem Graph mit Adjazenzmatrix A) von Knoten i zu Knoten j. Dann gibt es in dem von Matrix B beschriebenen Graphen G‘ einen Weg von i nach j mit Länge maximal $\frac{2}{3}k$.

Beweis

- Sei P ein Weg in G mit k Kanten.
- Ist k gerade, dann gibt es in G‘ einen Weg der Länge $k/2$, da man jeweils zwei aufeinanderfolgende Kanten von P durch eine Kante in G‘ ersetzen kann.
- Ist k ungerade, dann gibt es in G‘ einen Weg der Länge $\lceil k/2 \rceil$, da man bis auf die letzte Kante jeweils zwei aufeinanderfolgende Kanten von P durch eine Kante in G‘ ersetzen kann.
- Somit gilt für $k = 3$, dass die Länge des Weges in G‘ $2/3$ der Länge des Weges in G ist. Für $k > 3$ verkürzt sich die Weglänge entsprechend mehr.

Graphalgorithmen

Behauptung 62

Sei P ein Weg der Länge $k > 1$ in G (dem Graph mit Adjazenzmatrix A) von Knoten i zu Knoten j. Dann gibt es in dem von Matrix B beschriebenen Graphen G‘ einen Weg von i nach j mit Länge maximal $\frac{2}{3}k$.

Beweis

- Sei P ein Weg in G mit k Kanten.
- Ist k gerade, dann gibt es in G‘ einen Weg der Länge $k/2$, da man jeweils zwei aufeinanderfolgende Kanten von P durch eine Kante in G‘ ersetzen kann.
- Ist k ungerade, dann gibt es in G‘ einen Weg der Länge $\lceil k/2 \rceil$, da man bis auf die letzte Kante jeweils zwei aufeinanderfolgende Kanten von P durch eine Kante in G‘ ersetzen kann.
- Somit gilt für $k = 3$, dass die Länge des Weges in G‘ $2/3$ der Länge des Weges in G ist. Für $k > 3$ verkürzt sich die Weglänge entsprechend mehr.

Graphalgorithmen

Konsequenz aus Beh. 61 und 62

Wenn wir die Berechnung von $B \log_{3/2} n$ mal iterieren, haben wir die transitive Hülle berechnet

Graphalgorithmen

TransitiveHülle(A)

1. **for** $i \leftarrow 1$ **to** $\log_{3/2} n$ **do**
2. $\quad Z' \leftarrow A\,A + A$
3. $\quad$**for** $i \leftarrow 1$ **to** n **do**
4. $\quad\quad$**for** $j \leftarrow 1$ **to** n **do**
5. $\quad\quad\quad$**if** $\mathrm{z}'_{ij} > 0$ **then** $b_{ij} \leftarrow 1$ **else** $b_{ij} \leftarrow 0$
6. $\quad A \leftarrow B$
7. **return** A

Graphalgorithmen

TransitiveHülle(A)	Laufzeit
1. **for** $i \leftarrow 1$ **to** $\log_{3/2} n$ **do**	$\mathbf{O}(\log n)$
2. $Z' \leftarrow A\,A + A$	$\mathbf{O}(M(n) \cdot \log n)$
3. **for** $i \leftarrow 1$ **to** n **do**	$\mathbf{O}(n \log n)$
4. **for** $j \leftarrow 1$ **to** n **do**	$\mathbf{O}(n^2 \log n)$
5. **if** $z'_{ij} > 0$ **then** $b_{ij} \leftarrow 1$ **else** $b_{ij} \leftarrow 0$	$\mathbf{O}(n^2 \log n)$
6. $A \leftarrow B$	$\mathbf{O}(n^2 \log n)$
7. **return** A	$\mathbf{O}(1)$

$M(n)$: Zeit um zwei $n \times n$-Matrizen zu multiplizieren

Graphalgorithmen

Satz 63

Der Algorithmus TransitiveHülle berechnet die transitive Hülle eines Graphen G in $\mathbf{O}(M(n)\log n)$ Zeit, wobei $M(n)$ die Laufzeit zur Multiplikation zweier $n \times n$-Matrizen bezeichnet.