

Vorlesung 06: Graphentheorie

Grundlagen Operations Research

Prof. Dr. Knut Haase

Grundbegriffe

Netze überall



Vlček u. a. (2024)

Knoten, Kanten, Pfeile

Knoten, Kanten, Pfeile

- **Knotenmenge** $\mathcal{V}$: die Objekte (Orte, Stationen, ...).

Knoten, Kanten, Pfeile

- **Knotenmenge** $\mathcal{V}$: die Objekte (Orte, Stationen, ...).
- **Kante** $[i, j] \in \mathcal{E}$: ungerichtete Verbindung.

Knoten, Kanten, Pfeile

- **Knotenmenge** $\mathcal{V}$: die Objekte (Orte, Stationen, ...).
- **Kante** $[i, j] \in \mathcal{E}$: ungerichtete Verbindung.
- **Pfeil** $(i, j) \in \vec{\mathcal{E}}$: gerichtete Verbindung.

Knoten, Kanten, Pfeile

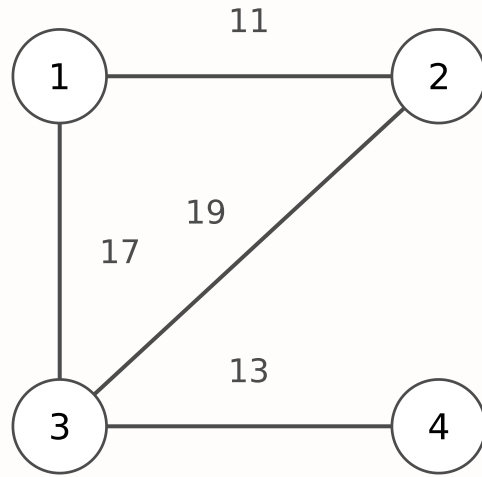
- **Knotenmenge** $\mathcal{V}$: die Objekte (Orte, Stationen, ...).
- **Kante** $[i, j] \in \mathcal{E}$: ungerichtete Verbindung.
- **Pfeil** $(i, j) \in \vec{\mathcal{E}}$: gerichtete Verbindung.
- **Bewertung** λ_{ij} : Kosten, Distanz oder Zeit einer Kante bzw. eines Pfeils.

Knoten, Kanten, Pfeile

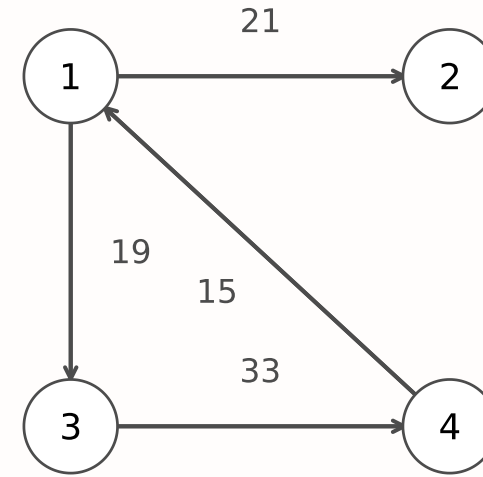
- **Knotenmenge** $\mathcal{V}$: die Objekte (Orte, Stationen, ...).
- **Kante** $[i, j] \in \mathcal{E}$: ungerichtete Verbindung.
- **Pfeil** $(i, j) \in \vec{\mathcal{E}}$: gerichtete Verbindung.
- **Bewertung** λ_{ij} : Kosten, Distanz oder Zeit einer Kante bzw. eines Pfeils.

Ein **schlichter** Graph hat keine Schlingen und keine parallelen Kanten; ein **Digraph** ist ein endlicher, schlichter, gerichteter Graph.

Gerichtet und ungerichtet



Ungerichteter Graph



Digraph

Spezielle Graphen

Spezielle Graphen

- **Zusammenhängend:** unter Vernachlässigung der Richtung sind alle Knoten direkt oder indirekt verbunden.

Spezielle Graphen

- **Zusammenhängend**: unter Vernachlässigung der Richtung sind alle Knoten direkt oder indirekt verbunden.
- **Baum**: zusammenhängend und **ohne** Kreis.

Spezielle Graphen

- **Zusammenhängend**: unter Vernachlässigung der Richtung sind alle Knoten direkt oder indirekt verbunden.
- **Baum**: zusammenhängend und **ohne** Kreis.
- **1-Baum**: zusammenhängend mit **genau einem** Kreis.

Spezielle Graphen

- **Zusammenhängend**: unter Vernachlässigung der Richtung sind alle Knoten direkt oder indirekt verbunden.
- **Baum**: zusammenhängend und **ohne** Kreis.
- **1-Baum**: zusammenhängend mit **genau einem** Kreis.

Der Kürzeste-Wege-Baum, den wir gleich berechnen, ist ein solcher **Baum**.

Speicherung

Warum kompakt speichern?

Eine Knoten-Knoten-Matrix braucht $|V|^2$ Einträge, bei **dünn besetzten** Netzen enorme Verschwendung.

Warum kompakt speichern?

Eine Knoten-Knoten-Matrix braucht $|V|^2$ Einträge, bei **dünn besetzten** Netzen enorme Verschwendung.

Besser: nur die **tatsächlich vorhandenen** Pfeile speichern (Vorwärtsstern).

Vorwärtsstern

Zwei Felder genügen:

Vorwärtsstern

Zwei Felder genügen:

- Ein **Pfeilfeld** B listet alle Pfeile (i_k, j_k) mit Bewertung $\lambda_{i_k j_k}$,
sortiert nach Startknoten.

Vorwärtsstern

Zwei Felder genügen:

- Ein **Pfeilfeld** B listet alle Pfeile (i_k, j_k) mit Bewertung $\lambda_{i_k j_k}$,
sortiert nach **Startknoten**.
- Ein **Knotenfeld** A speichert je Knoten i den Index k_i seines **ersten**
Pfeils in B .

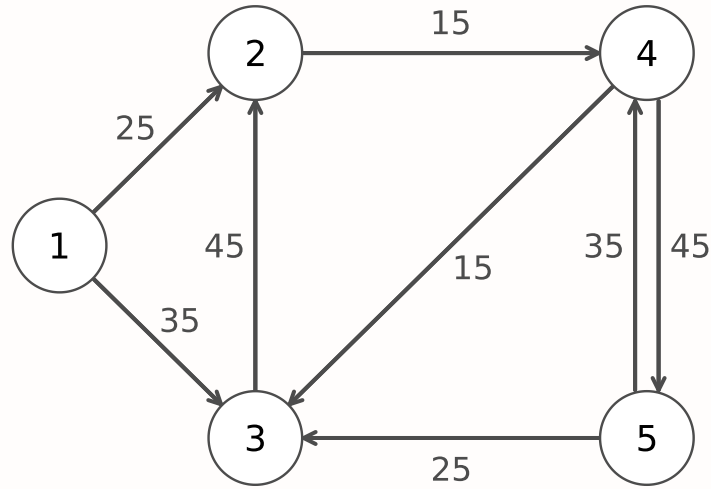
Vorwärtsstern

Zwei Felder genügen:

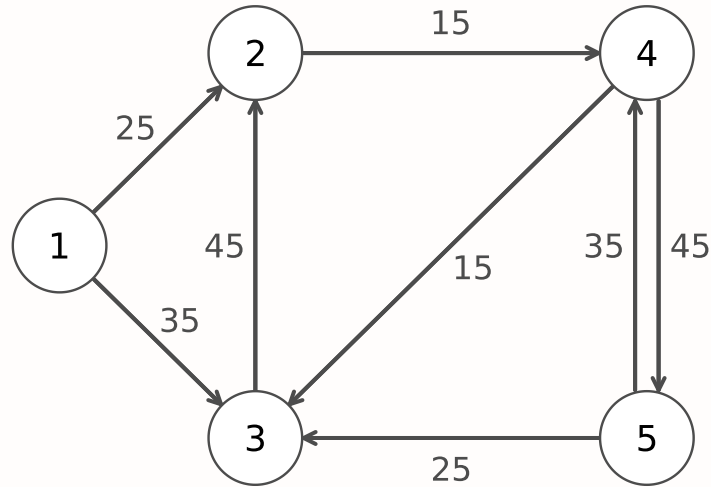
- Ein **Pfeilfeld** B listet alle Pfeile (i_k, j_k) mit Bewertung $\lambda_{i_k j_k}$,
sortiert nach Startknoten.
- Ein **Knotenfeld** A speichert je Knoten i den Index k_i seines **ersten**
Pfeils in B .

Die Nachfolger von i stehen dann zusammenhängend zwischen k_i
und $k_{i+1} - 1$. Der Speicherbedarf sinkt von $\mathcal{O}(|V|^2)$ auf $\mathcal{O}(|V| + |\vec{E}|)$.

Vorwärtsstern am Beispiel



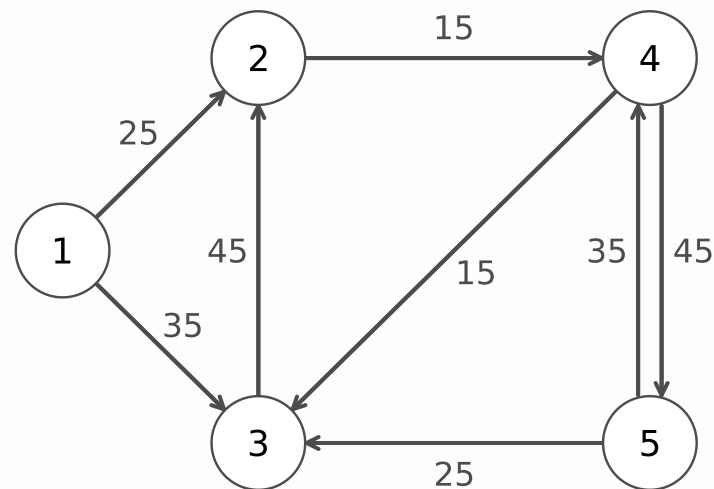
Vorwärtsstern am Beispiel



i	1	2	3	4	5	(6)
k_i	1	3	4	5	7	9

Knotenfeld A

Vorwärtsstern am Beispiel



i	1	2	3	4	5	(6)
k_i	1	3	4	5	7	9

Knotenfeld A

Das Pfeilfeld B listet darunter alle acht Pfeile, **nach Startknoten sortiert**.

k	1	2	3	4	5	6	7	8
(i_k, j_k)	(1,2)	(1,3)	(2,4)	(3,2)	(4,3)	(4,5)	(5,3)	(5,4)
$\lambda_{i_k j_k}$	25	35	15	45	15	45	25	35

Nachfolger nachschlagen

k	1	2	3	4	5	6	7	8
(i_k, j_k)	(1,2)	(1,3)	(2,4)	(3,2)	(4,3)	(4,5)	(5,3)	(5,4)
$\lambda_{i_k j_k}$	25	35	15	45	15	45	25	35

Frage: Wie finden Sie die Nachfolger von Knoten 4?

Nachfolger nachschlagen

k	1	2	3	4	5	6	7	8
(i_k, j_k)	(1,2)	(1,3)	(2,4)	(3,2)	(4,3)	(4,5)	(5,3)	(5,4)
$\lambda_{i_k j_k}$	25	35	15	45	15	45	25	35

Frage: Wie finden Sie die Nachfolger von Knoten 4?

Knotenfeld A : $k_4 = 5$ und $k_5 - 1 = 6 \Rightarrow$ die Pfeile (4,3) und (4,5),
ohne die Matrix zu durchsuchen.

Der Rückwärtsstern

Für **Vorgänger**-Fragen (z. B. in der Netzplantechnik) sortiert man dieselben Pfeile nach dem **Zielknoten**:

Der Rückwärtsstern

Für **Vorgänger**-Fragen (z. B. in der Netzplantechnik) sortiert man dieselben Pfeile nach dem **Zielknoten**:

- Ein **Pfeilfeld** C listet in Zeile n die Nummer $\tilde{k}_n$ eines Pfeils aus B ,
sortiert nach **Zielknoten**.

Der Rückwärtsstern

Für **Vorgänger**-Fragen (z. B. in der Netzplantechnik) sortiert man dieselben Pfeile nach dem **Zielknoten**:

- Ein **Pfeilfeld** C listet in Zeile n die Nummer $\tilde{k}_n$ eines Pfeils aus B , sortiert nach **Zielknoten**.
- Ein **Knotenfeld** D speichert je Knoten j den Index n_j seines ersten Eintrags in C .

Der Rückwärtsstern

Für **Vorgänger**-Fragen (z. B. in der Netzplantechnik) sortiert man dieselben Pfeile nach dem **Zielknoten**:

- Ein **Pfeilfeld** C listet in Zeile n die Nummer $\tilde{k}_n$ eines Pfeils aus B ,
sortiert nach **Zielknoten**.
- Ein **Knotenfeld** D speichert je Knoten j den Index n_j seines
ersten Eintrags in C .

Die Vorgänger von j stehen dann zwischen n_j und $n_{j+1} - 1$ in C .

Rückwärtsstern am Beispiel

n	1	2	3	4	5	6	7	8
$\tilde{k}_n$	1	4	2	5	7	3	8	6

Pfeilfeld C

Rückwärtsstern am Beispiel

n	1	2	3	4	5	6	7	8
$\tilde{k}_n$	1	4	2	5	7	3	8	6

Pfeilfeld C

j	1	2	3	4	5	(6)
n_j	1	1	3	6	8	9

Knotenfeld D

Rückwärtsstern am Beispiel

n	1	2	3	4	5	6	7	8
$\tilde{k}_n$	1	4	2	5	7	3	8	6

Pfeilfeld C

j	1	2	3	4	5	(6)
n_j	1	1	3	6	8	9

Knotenfeld D

Die Vorgänger von Knoten 3: $n_3 = 3$ und $n_4 - 1 = 5 \Rightarrow$ die Pfeile 2, 5, 7, also (1, 3), (4, 3) und (5, 3).

Kürzeste Wege

Aufgabenstellung

Gesucht: die **kürzesten Wege** von einem Startknoten q (**Quelle**) zu allen anderen Knoten.

Aufgabenstellung

Gesucht: die **kürzesten Wege** von einem Startknoten q (**Quelle**) zu allen anderen Knoten.

- λ_{ij} : Bewertung des Pfeils (i, j) (Zeit, Distanz, Kosten).

Aufgabenstellung

Gesucht: die **kürzesten Wege** von einem Startknoten q (**Quelle**) zu allen anderen Knoten.

- λ_{ij} : Bewertung des Pfeils (i, j) (Zeit, Distanz, Kosten).
- $\mathcal{N}(i)$: Menge der **Nachfolger** von i .

Aufgabenstellung

Gesucht: die **kürzesten Wege** von einem Startknoten q (**Quelle**) zu allen anderen Knoten.

- λ_{ij} : Bewertung des Pfeils (i, j) (Zeit, Distanz, Kosten).
- $\mathcal{N}(i)$: Menge der **Nachfolger** von i .
- $D[j]$: bisher kürzeste Entfernung von q nach j ; Start: $D[q] = 0$, sonst ∞ .

Aufgabenstellung

Gesucht: die **kürzesten Wege** von einem Startknoten q (**Quelle**) zu allen anderen Knoten.

- λ_{ij} : Bewertung des Pfeils (i, j) (Zeit, Distanz, Kosten).
- $\mathcal{N}(i)$: Menge der **Nachfolger** von i .
- $D[j]$: bisher kürzeste Entfernung von q nach j ; Start: $D[q] = 0$, sonst ∞ .
- $R[j]$: bester **Vorgänger** von j (zum Rekonstruieren des Weges).

Der FIFO-Algorithmus

Idee

Knoten werden in der Reihenfolge einer **FIFO-Warteschlange** abgearbeitet. Findet man einen kürzeren Weg zu j , wird $D[j]$ aktualisiert und j hinten angestellt: solange, bis die Schlange leer ist.

Der FIFO-Algorithmus



Idee

Knoten werden in der Reihenfolge einer **FIFO-Warteschlange** abgearbeitet. Findet man einen kürzeren Weg zu j , wird $D[j]$ aktualisiert und j hinten angestellt: solange, bis die Schlange leer ist.

Ein Knoten steht **höchstens einmal** gleichzeitig in der Schlange, kann aber insgesamt **mehrfach** bearbeitet werden (Label-Correcting).

Iterative Vorgehensweise

Schlange mit Kopf H und Ende T ; Start $H = T = q$.

Iterative Vorgehensweise

Schlange mit Kopf H und Ende T ; Start $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?

Iterative Vorgehensweise

Schlange mit Kopf H und Ende T ; Start $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?
- Falls ja: $D[j] := D[H] + \lambda_{H,j}$, $R[j] := H$, und j hinten einreihen (falls nicht bereits enthalten).

Iterative Vorgehensweise

Schlange mit Kopf H und Ende T ; Start $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?
- Falls ja: $D[j] := D[H] + \lambda_{H,j}$, $R[j] := H$, und j hinten einreihen (falls nicht bereits enthalten).
- Ist $H = T$: Schlange leer $\Rightarrow$ **Abbruch**.

Iterative Vorgehensweise

Schlange mit Kopf H und Ende T ; Start $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?
- Falls ja: $D[j] := D[H] + \lambda_{H,j}$, $R[j] := H$, und j hinten einreihen (falls nicht bereits enthalten).
- Ist $H = T$: Schlange leer $\Rightarrow$ **Abbruch**.
- Sonst H auf den nächsten Knoten der Liste setzen.

Die Schlange im Feld S

Die markierten Knoten stehen in einem Feld S , ein Eintrag je Knoten:

Die Schlange im Feld S

Die markierten Knoten stehen in einem Feld S , ein Eintrag je Knoten:

- $S[j]$: der Knoten, der in der Schlange **auf j folgt**.

Die Schlange im Feld S

Die markierten Knoten stehen in einem Feld S , ein Eintrag je Knoten:

- $S[j]$: der Knoten, der in der Schlange **auf j folgt**.
- j hinten einreihen: $S[T] := j$ und $T := j$.

Die Schlange im Feld S

Die markierten Knoten stehen in einem Feld S , ein Eintrag je Knoten:

- $S[j]$: der Knoten, der in der Schlange **auf j folgt**.
- j hinten einreihen: $S[T] := j$ und $T := j$.
- Nächster Kopf: $H := S[H]$.

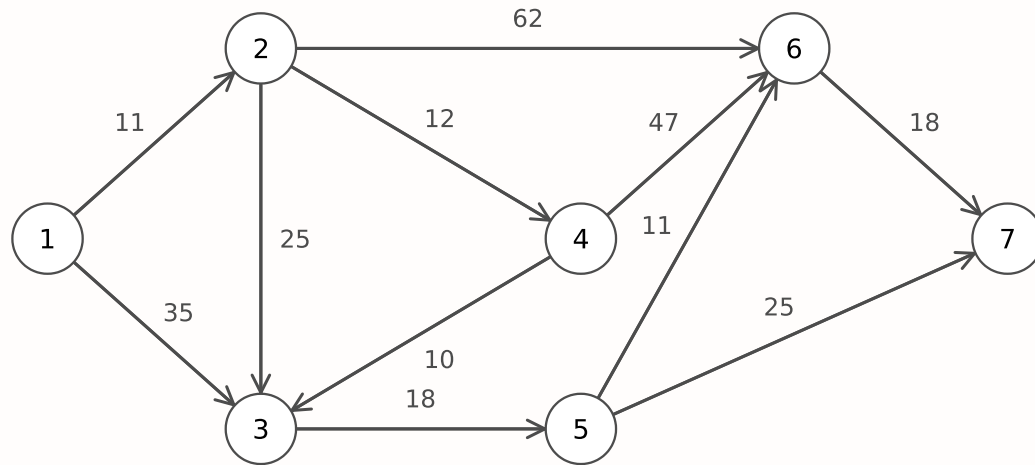
Die Schlange im Feld S

Die markierten Knoten stehen in einem Feld S , ein Eintrag je Knoten:

- $S[j]$: der Knoten, der in der Schlange **auf j folgt**.
- j hinten einreihen: $S[T] := j$ und $T := j$.
- Nächster Kopf: $H := S[H]$.

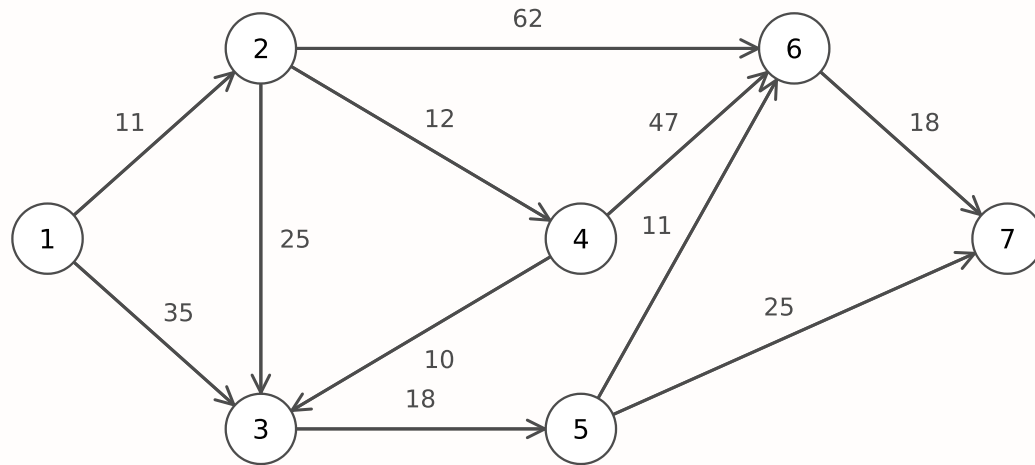
Die **FIFO-Liste L** hält den ganzen Lauf fest: Abgearbeitete Knoten werden ~~gestrichen~~, die übrigen bilden die aktuelle Schlange.

Beispiel: Busliniennetz



Kürzeste Wege von Knoten $q = 1$ zu allen anderen.

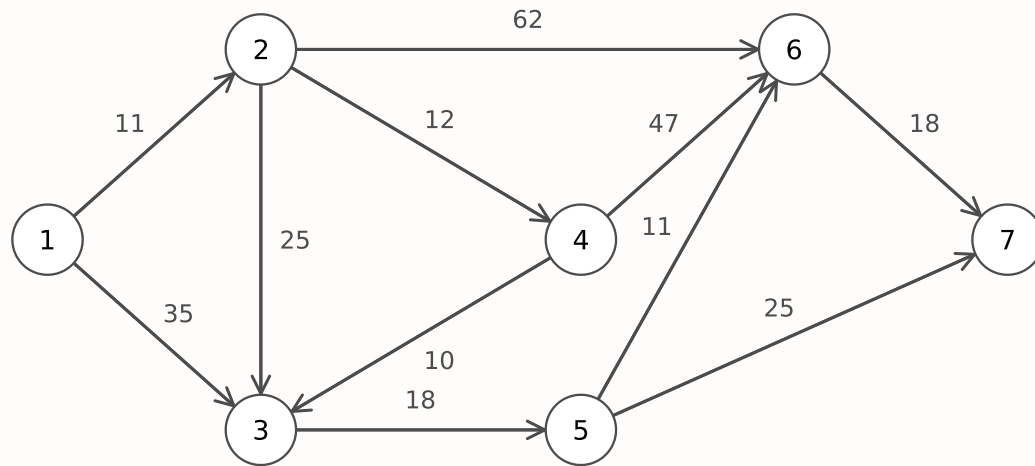
Beispiel: Busliniennetz



Kürzeste Wege von Knoten $q = 1$ zu allen anderen.

Frage: Welchen Weg vermuten Sie von 1 nach 3: direkt oder über Umwege?

Beispiel: Busliniennetz



Kürzeste Wege von Knoten $q = 1$ zu allen anderen.

Frage: Welchen Weg vermuten Sie von 1 nach 3: direkt oder über Umwege?

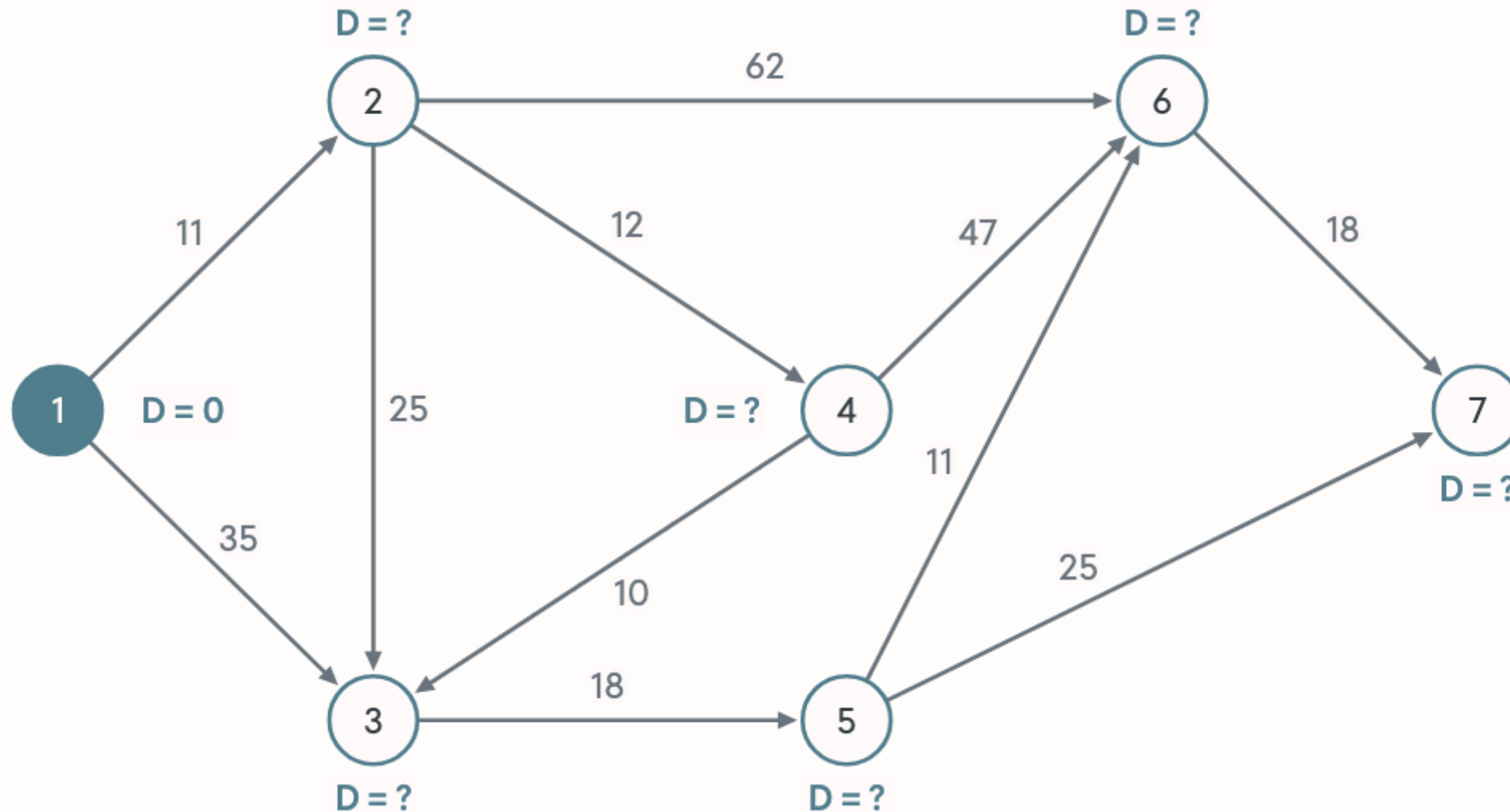


Gleich ausprobieren

Auf der nächsten Folie suchen Sie die kürzesten Wege von Knoten 1 zu **allen** anderen Knoten. Für jeden Knoten wählen Sie den Pfeil, über den er erreicht wird. Stimmt Ihre Vermutung zu Knoten 3?

Sie sind dran

Tippen Sie auf einen Pfeil: Über ihn wird sein Zielknoten von Knoten 1 aus erreicht. An jedem Knoten steht seine Entfernung. Machen Sie die Summe aller Entfernungen möglichst klein.



Eigener Plan 0

Noch ohne Pfeil:
Knoten 2, 3, 4, 5, 6
und 7

Optimieren

Reset

Neu

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	∞	∞	∞	∞	∞	∞
$R[j]$	–	–	–	–	–	–	–

Start: $H = T = q = 1$.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	35	∞	∞	∞	∞
$R[j]$	–	1	1	–	–	–	–

Knoten 1: beide Nachfolger erhalten Marken und stellen sich hinten an.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	35	23	∞	73	∞
$R[j]$	–	1	1	2	–	2	–

Knoten 2: 4 und 6 sind neu. Für 3 gilt $11 + 25 = 36 > 35$: **keine** Korrektur.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	35	23	53	73	∞
$R[j]$	–	1	1	2	3	2	–

Knoten 3: Knoten 5 erhält $35 + 18 = 53$.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	53	70	∞
$R[j]$	–	1	4	2	3	4	–

Knoten 4 **korrigiert**: $23 + 10 = 33 < 35$. Knoten 3 stellt sich **erneut** an!
Auch $D[6] = 70$ (6 wartet bereits).

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	53	70	88
$R[j]$	–	1	4	2	3	4	6

Knoten 6: Knoten 7 erhält $70 + 18 = 88$ und stellt sich hinten an.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	53	64	78
$R[j]$	–	1	4	2	3	5	5

Knoten 5 **korrigiert** 6 ($53 + 11 = 64 < 70$) und 7 ($53 + 25 = 78 < 88$). 6 stellt sich **erneut** an, 7 wartet.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	64	78
$R[j]$	–	1	4	2	3	5	5

Knoten 3 **korrigiert** 5 ($33 + 18 = 51 < 53$). 5 stellt sich **erneut** an.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	64	78
$R[j]$	–	1	4	2	3	5	5

Knoten 7 hat **keine** Nachfolger, es ändert sich nichts.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	64	78
$R[j]$	–	1	4	2	3	5	5

Knoten 6 prüft 7: $64 + 18 = 82 > 78$, also **keine** Korrektur.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	62	76
$R[j]$	–	1	4	2	3	5	5

Knoten 5 **korrigiert** 6 ($51 + 11 = 62 < 64$) und 7 ($51 + 25 = 76 < 78$).
Beide stellen sich **erneut** an.

Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	62	76
$R[j]$	–	1	4	2	3	5	5

Knoten 6 prüft 7: $62 + 18 = 80 > 76$, also **keine** Korrektur.

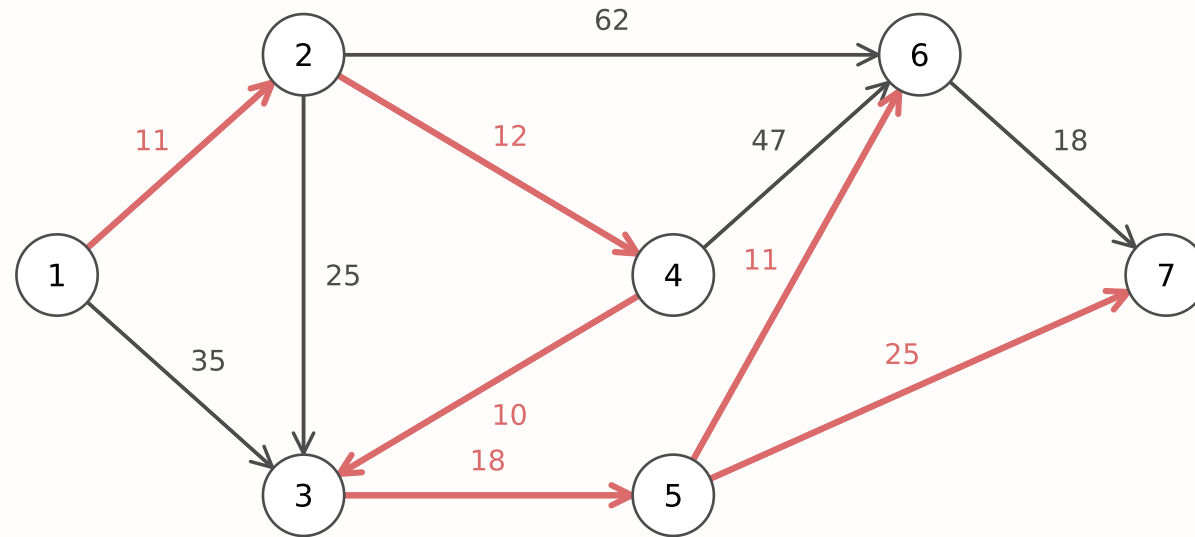
Der Ablauf im Detail

Jetzt rechnet FIFO dasselbe, Knoten für Knoten.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	62	76
$R[j]$	–	1	4	2	3	5	5

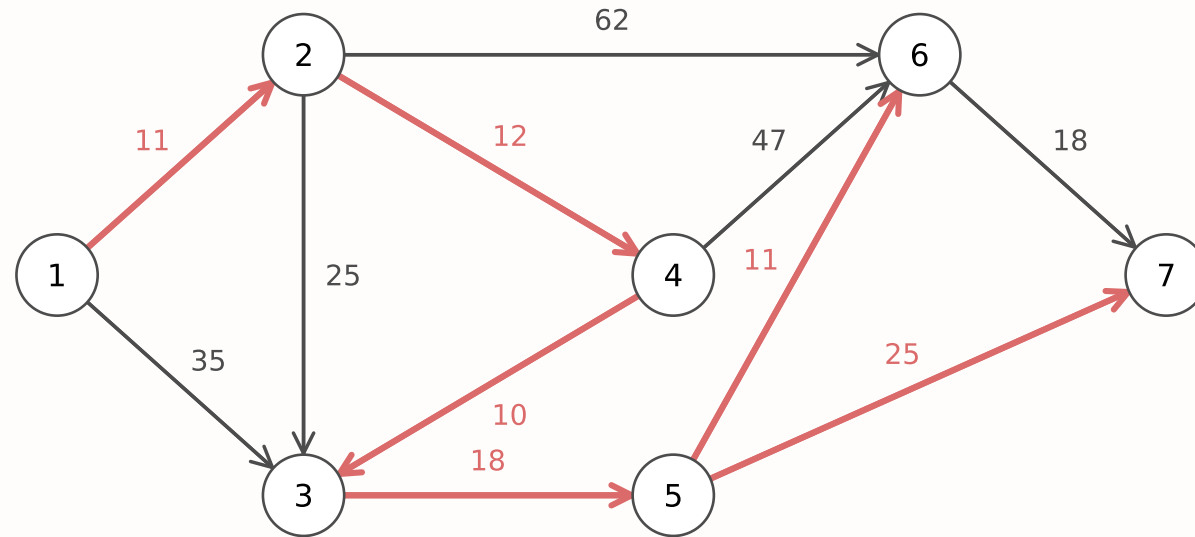
Knoten 7 hat **keine** Nachfolger. Wegen $H = T$ ist die Schlange **leer**:
Abbruch.

Ergebnis als Baum



Die roten Pfeile bilden den **Kürzeste-Wege-Baum**.

Ergebnis als Baum



Die roten Pfeile bilden den **Kürzeste-Wege-Baum**.

! Kürzeste Distanzen

$D = (0, 11, 33, 23, 51, 62, 76)$

FIFO-LIFO

Der FIFO-LIFO-Algorithmus

Idee

Ein Knoten in der Liste löst Korrekturen bei seinen Nachfolgern aus, bei deren Nachfolgern usw. Muss er **erneut** hinein, lohnt es nicht, erst die Knoten abzuarbeiten, die er vorher hineingebracht hat: Sie werden voraussichtlich wieder korrigiert.

Der FIFO-LIFO-Algorithmus

Idee

Ein Knoten in der Liste löst Korrekturen bei seinen Nachfolgern aus, bei deren Nachfolgern usw. Muss er **erneut** hinein, lohnt es nicht, erst die Knoten abzuarbeiten, die er vorher hineingebracht hat: Sie werden voraussichtlich wieder korrigiert.

Wer zum **wiederholten** Mal in die Liste kommt, wird deshalb **vorne** hinter dem Kopf eingereiht: das **FIFO-LIFO-Prinzip**.

Vorne oder hinten?

Frage: Woran erkennen Sie, dass ein Knoten schon einmal in der Liste stand?

Vorne oder hinten?

Frage: Woran erkennen Sie, dass ein Knoten schon einmal in der Liste stand?

! Erkennungsregel

Bis auf q starten alle Knoten mit $D[j] = \infty$. Ist $D[j]$ **vor** der Korrektur schon **endlich**, stand j schon in der Liste. Steht er nicht mehr in der Schlange, ist er bereits bearbeitet: Er kommt **vorne** hin.

Vorne oder hinten?

Frage: Woran erkennen Sie, dass ein Knoten schon einmal in der Liste stand?

! Erkennungsregel

Bis auf q starten alle Knoten mit $D[j] = \infty$. Ist $D[j]$ **vor** der Korrektur schon **endlich**, stand j schon in der Liste. Steht er nicht mehr in der Schlange, ist er bereits bearbeitet: Er kommt **vorne** hin.

Ein **zusätzliches Feld** ginge auch, ist so aber **nicht** nötig.

Vorne einreihen im Feld S

Ausschnitt aus einem Lauf mit sieben Knoten:

j	1	2	3	4	5	6	7
$S[j]$	–	–	6	–	3	4	–

Kopf $H = 5$, Ende $T = 4$. Knoten 1 ist zum **zweiten** Mal aufzunehmen.
 $\bar{L} = (5, 3, 6, 4)$

Vorne einreihen im Feld S

Ausschnitt aus einem Lauf mit sieben Knoten:

j	1	2	3	4	5	6	7
$S[j]$	3	–	6	–	1	4	–

$S[1] := S[5] = 3$, dann $S[5] := 1$ (allgemein $S[j] := S[H]$ und $S[H] := j$).
 $L = (5, 1, 3, 6, 4)$

Ablauf von FIFO-LIFO

Start wie bei FIFO: $H = T = q$.

Ablauf von FIFO-LIFO

Start wie bei FIFO: $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?

Ablauf von FIFO-LIFO

Start wie bei FIFO: $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?
- Falls ja: $D[j] := D[H] + \lambda_{H,j}$, $R[j] := H$; steht j nicht in der Schlange, wird er eingereiht:

Ablauf von FIFO-LIFO

Start wie bei FIFO: $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?
- Falls ja: $D[j] := D[H] + \lambda_{H,j}$, $R[j] := H$; steht j nicht in der Schlange, wird er eingereiht:
- **hinten**, falls bisher $D[j] = \infty$: $S[T] := j$, $T := j$;

Ablauf von FIFO-LIFO

Start wie bei FIFO: $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?
- Falls ja: $D[j] := D[H] + \lambda_{H,j}$, $R[j] := H$; steht j nicht in der Schlange, wird er eingereiht:
- **hinten**, falls bisher $D[j] = \infty$: $S[T] := j$, $T := j$;
- sonst **vorne** hinter H : $S[j] := S[H]$, $S[H] := j$ (ist $H = T$, auch $T := j$).

Ablauf von FIFO-LIFO

Start wie bei FIFO: $H = T = q$.

- Für jeden Nachfolger $j \in \mathcal{N}(H)$: ist $D[j] > D[H] + \lambda_{H,j}$?
- Falls ja: $D[j] := D[H] + \lambda_{H,j}$, $R[j] := H$; steht j nicht in der Schlange, wird er eingereiht:
- **hinten**, falls bisher $D[j] = \infty$: $S[T] := j$, $T := j$;
- sonst **vorne** hinter H : $S[j] := S[H]$, $S[H] := j$ (ist $H = T$, auch $T := j$).
- Ist $H = T$: **Abbruch**; sonst $H := S[H]$.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	∞	∞	∞	∞	∞	∞
$R[j]$	–	–	–	–	–	–	–

Start: $H = T = q = 1$.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	35	∞	∞	∞	∞
$R[j]$	–	1	1	–	–	–	–

Knoten 1: beide Nachfolger erhalten **erstmal**s Marken und stellen sich hinten an.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	35	23	∞	73	∞
$R[j]$	–	1	1	2	–	2	–

Knoten 2: 4 und 6 sind neu und kommen hinten hin. Für 3 gilt $11 + 25 = 36 > 35$: **keine** Korrektur.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	35	23	53	73	∞
$R[j]$	–	1	1	2	3	2	–

Knoten 3: Knoten 5 erhält $35 + 18 = 53$ und stellt sich hinten an.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	53	70	∞
$R[j]$	–	1	4	2	3	4	–

Knoten 4 **korrigiert** 3 ($23 + 10 = 33 < 35$). $D[3]$ war endlich: 3 kommt **vorne** hin. Auch $D[6] = 70$ (6 wartet).

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	70	∞
$R[j]$	–	1	4	2	3	4	–

Knoten 3 ist **sofort** wieder dran und korrigiert 5 ($33 + 18 = 51 < 53$), bevor 5 überhaupt bearbeitet wird.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	70	88
$R[j]$	–	1	4	2	3	4	6

Knoten 6: Knoten 7 erhält **erstmal**s $70 + 18 = 88$ und stellt sich hinten an.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	62	76
$R[j]$	–	1	4	2	3	5	5

Knoten 5 **korrigiert** 6 ($51 + 11 = 62 < 70$) und 7 ($51 + 25 = 76 < 88$). 6 kommt **vorne** hin, 7 wartet.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	62	76
$R[j]$	–	1	4	2	3	5	5

Knoten 6 prüft 7: $62 + 18 = 80 > 76$, also **keine** Korrektur.

FIFO-LIFO am Busliniennetz

Dasselbe Netz, aber wer **erneut** markiert wird, kommt **vorne** hin.

j	1	2	3	4	5	6	7
$D[j]$	0	11	33	23	51	62	76
$R[j]$	–	1	4	2	3	5	5

Knoten 7 hat **keine** Nachfolger. Wegen $H = T$ ist die Schlange **leer**:
Abbruch.

Varianten des Verfahrens

Varianten des Verfahrens

- **FIFO-LIFO** braucht am Busliniennetz 9 statt 12 Iterationen bei gleichen Distanzen: 5 und 7 kommen nur **einmal** dran. Das spart nur, wenn Knoten **erneut** in die Schlange kommen; sonst sind beide gleich schnell.

Varianten des Verfahrens

- **FIFO-LIFO** braucht am Busliniennetz 9 statt 12 Iterationen bei gleichen Distanzen: 5 und 7 kommen nur **einmal** dran. Das spart nur, wenn Knoten **erneut** in die Schlange kommen; sonst sind beide gleich schnell.
- **FIFO-FIFO**: mehrere erneut aufzunehmende Knoten behalten ihre Reihenfolge, eingefügt hinter dem zuletzt **eingefügten** Knoten.

Varianten des Verfahrens

- **FIFO-LIFO** braucht am Busliniennetz 9 statt 12 Iterationen bei gleichen Distanzen: 5 und 7 kommen nur **einmal** dran. Das spart nur, wenn Knoten **erneut** in die Schlange kommen; sonst sind beide gleich schnell.
- **FIFO-FIFO**: mehrere erneut aufzunehmende Knoten behalten ihre Reihenfolge, eingefügt hinter dem zuletzt **eingefügten** Knoten.

Beide Varianten reduzieren **überflüssige** Aktualisierungen.

Netzwerkfluss

Kürzeste Wege als LP

Kürzeste Wege lassen sich als **Netzwerkflussproblem** formulieren:
Wir schicken von q eine Einheit zu **jedem** anderen Knoten.

Kürzeste Wege als LP

Kürzeste Wege lassen sich als **Netzwerkflussproblem** formulieren:
Wir schicken von q eine Einheit zu **jedem** anderen Knoten.

- c_{ij} : Kosten pro Mengeneinheit auf Pfeil (i, j) .

Kürzeste Wege als LP

Kürzeste Wege lassen sich als **Netzwerkflussproblem** formulieren:
Wir schicken von q eine Einheit zu **jedem** anderen Knoten.

- c_{ij} : Kosten pro Mengeneinheit auf Pfeil (i, j) .
- X_{ij} : Transportmenge (Fluss) auf Pfeil (i, j) .

Kürzeste Wege als LP

Kürzeste Wege lassen sich als **Netzwerkflussproblem** formulieren:
Wir schicken von q eine Einheit zu **jedem** anderen Knoten.

- c_{ij} : Kosten pro Mengeneinheit auf Pfeil (i, j) .
- X_{ij} : Transportmenge (Fluss) auf Pfeil (i, j) .

Ist nur der kürzeste Weg zu **einem** Zielknoten s (**Senke**) gesucht, fließt genau eine Einheit von q nach s .

Das Modell

$$\text{Minimiere } F = \sum_{(i,j) \in \vec{\mathcal{E}}} c_{ij} X_{ij}$$

Flusserhaltung (Zufluss – Abfluss = Bedarf) für jeden Knoten i :

$$\sum_{j: (j,i)} X_{ji} - \sum_{j: (i,j)} X_{ij} = \begin{cases} 1 - |\mathcal{V}| & i = q \\ 1 & \text{sonst} \end{cases}$$

Das Modell am Beispiel

$$\text{Minimiere } F = 11X_{12} + 35X_{13} + 25X_{23} + \dots + 18X_{67}$$

Das Modell am Beispiel

Minimiere $F = 11X_{12} + 35X_{13} + 25X_{23} + \dots + 18X_{67}$

Flusserhaltung, links die Knoten 1 bis 4, rechts 5 bis 7:

$$-X_{12} - X_{13} = -6$$

$$X_{12} - X_{23} - X_{24} - X_{26} = 1$$

$$X_{13} + X_{23} + X_{43} - X_{35} = 1$$

$$X_{24} - X_{43} - X_{46} = 1$$

$$X_{35} - X_{56} - X_{57} = 1$$

$$X_{26} + X_{46} + X_{56} - X_{67} = 1$$

$$X_{57} + X_{67} = 1$$

$$X_{12}, X_{13}, \dots, X_{67} \geq 0$$

Kontrolle mit Julia

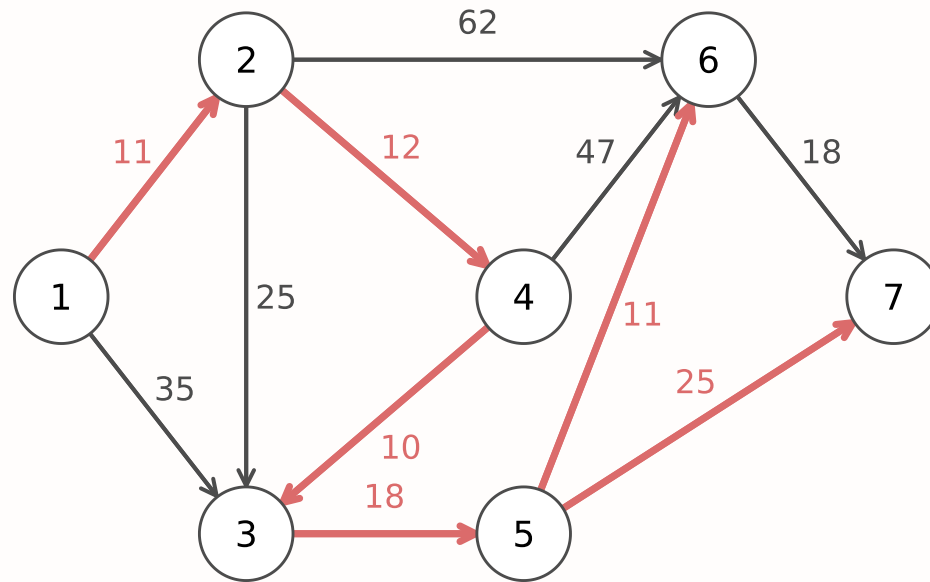
```
1 using JuMP, HiGHS
2
3 kanten = [(1,2,11),(1,3,35),(2,3,25),(2,4,12),(4,3,10),(3,5,18),
4           (2,6,62),(4,6,47),(5,6,11),(5,7,25),(6,7,18)]
5 V, q = 1:7, 1
6
7 modell = Model(HiGHS.Optimizer); set_silent(modell)
8 @variable(modell, X[e in kanten] >= 0) # Fluss je Pfeil
9 @objective(modell, Min, sum(e[3] * X[e] for e in kanten)) # e[3]: Kosten
10
11 for i in V # Flusserhaltung
12     zu = sum(X[e] for e in kanten if e[2] == i; init = 0.0)
13     ab = sum(X[e] for e in kanten if e[1] == i; init = 0.0)
14     @constraint(modell, zu - ab == (i == q ? 1 - length(V) : 1))
15 end
16 optimize!(modell)
17 println("Gesamtkosten des Flusses: F* = ", objective_value(modell))
```

Kontrolle mit Julia

```
1 using JuMP, HiGHS
2
3 kanten = [(1,2,11),(1,3,35),(2,3,25),(2,4,12),(4,3,10),(3,5,18),
4           (2,6,62),(4,6,47),(5,6,11),(5,7,25),(6,7,18)]
5 V, q = 1:7, 1
6
7 modell = Model(HiGHS.Optimizer); set_silent(modell)
8 @variable(modell, X[e in kanten] >= 0) # Fluss je Pfeil
9 @objective(modell, Min, sum(e[3] * X[e] for e in kanten)) # e[3]: Kosten
10
11 for i in V # Flusserhaltung
12     zu = sum(X[e] for e in kanten if e[2] == i; init = 0.0)
13     ab = sum(X[e] for e in kanten if e[1] == i; init = 0.0)
14     @constraint(modell, zu - ab == (i == q ? 1 - length(V) : 1))
15 end
16 optimize!(modell)
17 println("Gesamtkosten des Flusses: F* = ", objective_value(modell))
```

Gesamtkosten des Flusses: F* = 256.0

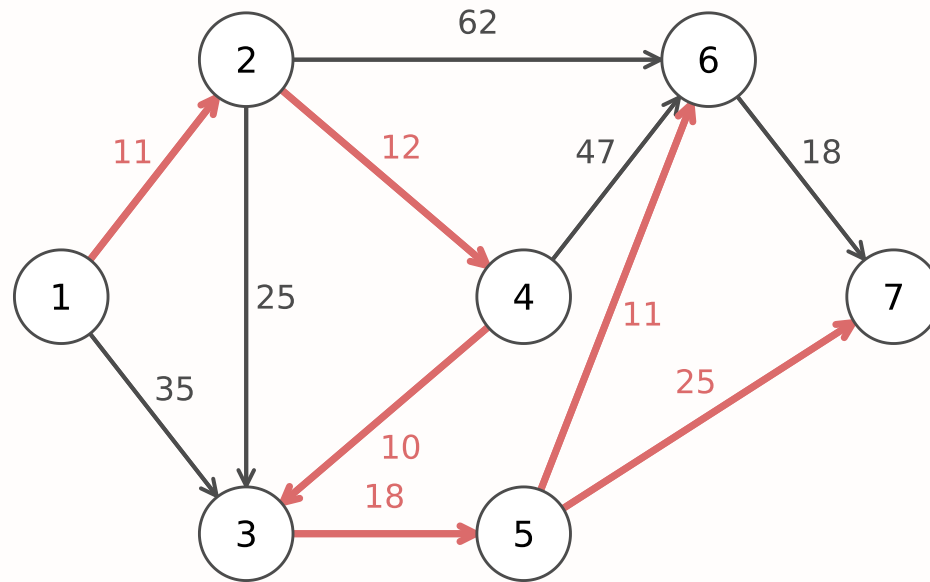
Der optimale Fluss



Der Fluss nutzt genau den **Kürzeste-Wege-Baum**:

$$X_{12} = 6, X_{24} = 5, X_{43} = 4, \\ X_{35} = 3, X_{56} = 1, X_{57} = 1.$$

Der optimale Fluss



Der Fluss nutzt genau den **Kürzeste-Wege-Baum**:

$$X_{12} = 6, X_{24} = 5, X_{43} = 4, \\ X_{35} = 3, X_{56} = 1, X_{57} = 1.$$

! **Schöner Zusammenhang**

$$F^* = 256 = \sum_j D[j]$$

Die Flusskosten sind die **Summe aller kürzesten Entfernungen**.

Zusammenfassung

Das Wichtigste auf einen Blick

Das Wichtigste auf einen Blick

- Ein **Graph** besteht aus Knoten und (gerichteten) Kanten mit Bewertungen.

Das Wichtigste auf einen Blick

- Ein **Graph** besteht aus Knoten und (gerichteten) Kanten mit Bewertungen.
- Die **kompakte Speicherung** (Vorwärtsstern) spart bei dünnen Netzen enorm Speicher.

Das Wichtigste auf einen Blick

- Ein **Graph** besteht aus Knoten und (gerichteten) Kanten mit Bewertungen.
- Die **kompakte Speicherung** (Vorwärtsstern) spart bei dünnen Netzen enorm Speicher.
- Der **FIFO-Algorithmus** bestimmt kürzeste Wege von q zu allen Knoten (Label-Correcting).

Das Wichtigste auf einen Blick

- Ein **Graph** besteht aus Knoten und (gerichteten) Kanten mit Bewertungen.
- Die **kompakte Speicherung** (Vorwärtsstern) spart bei dünnen Netzen enorm Speicher.
- Der **FIFO-Algorithmus** bestimmt kürzeste Wege von q zu allen Knoten (Label-Correcting).
- **FIFO-LIFO** reiht erneut aufzunehmende Knoten **vorne** ein und spart so Iterationen, wenn Marken korrigiert werden.

Das Wichtigste auf einen Blick

- Ein **Graph** besteht aus Knoten und (gerichteten) Kanten mit Bewertungen.
- Die **kompakte Speicherung** (Vorwärtsstern) spart bei dünnen Netzen enorm Speicher.
- Der **FIFO-Algorithmus** bestimmt kürzeste Wege von q zu allen Knoten (Label-Correcting).
- **FIFO-LIFO** reiht erneut aufzunehmende Knoten **vorne** ein und spart so Iterationen, wenn Marken korrigiert werden.
- Kürzeste Wege sind ein Spezialfall des **Netzwerkflussproblems**, lösbar als LP.

Literatur

Literaturverzeichnis

- Domschke, Wolfgang, Andreas Drexl, Robert Klein, und Armin Scholl. 2015. *Einführung in Operations Research*. 9. Aufl. Springer Gabler. <https://doi.org/10.1007/978-3-662-48216-2>.
- Vlček, Tobias, Knut Haase, Malte Fliedner, und Tobias Cors. 2024. „Police Service District Planning“. *OR Spectrum* 46 (4): 1029–61. <https://doi.org/10.1007/s00291-024-00745-3>.