Vorlesung 06: Graphentheorie

Grundlagen Operations Research

Grundbegriffe

Netze überall

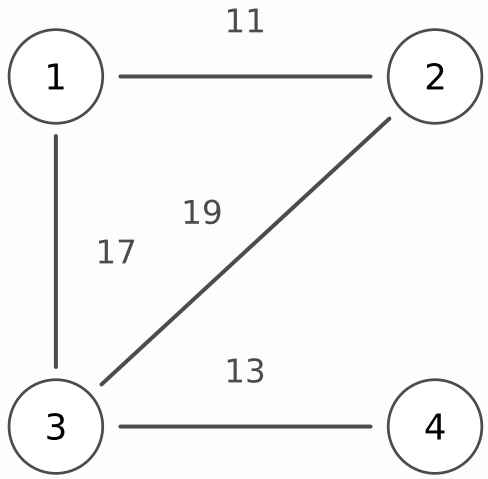
T. Vlćek, K. Haase, M. Fliedner, und T. Cors [1]

Knoten, Kanten, Pfeile

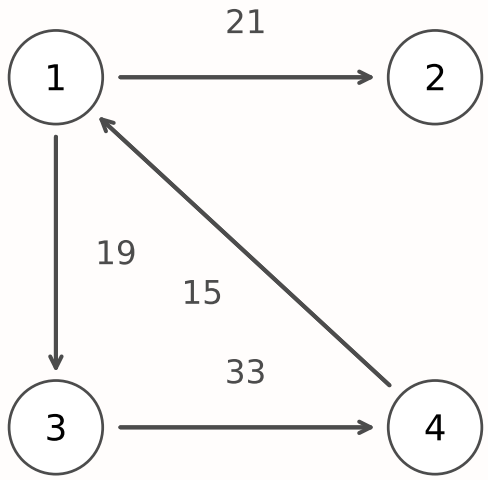
- **Knotenmenge** $\mathcal{V}$: die Objekte (Orte, Stationen, …).
- **Kante** $[i, j] \in \mathcal{E}$: ungerichtete Verbindung.
- **Pfeil** $(i, j) \in \vec{\mathcal{E}}$: gerichtete Verbindung.
- **Bewertung** λ_{ij}: Kosten, Distanz oder Zeit einer Kante bzw. eines Pfeils.

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

- **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**.

Kompakte Speicherung

Warum kompakt speichern?

Eine Knoten-Knoten-Matrix braucht $\mid \mathcal{V} \mid^2$ Einträge, bei **dünn besetzten** Netzen enorme Verschwendung.

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

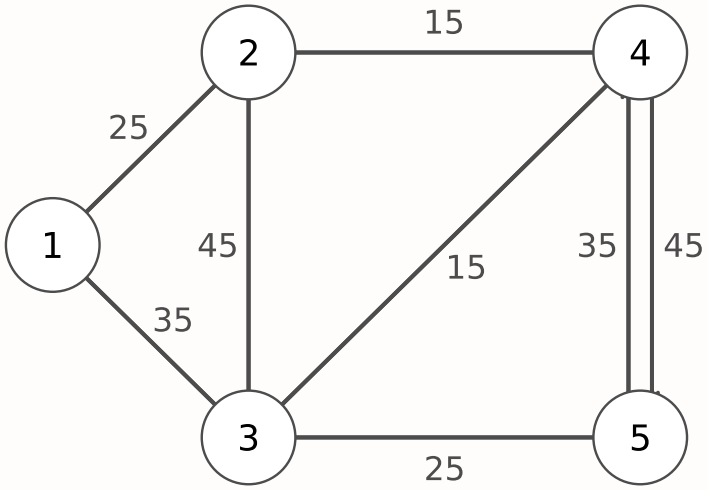
Vorwärtsstern

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 $\mathcal{O}(\mid \mathcal{V} \mid^2)$ auf $\mathcal{O}(\mid \mathcal{V} \mid + \mid \vec{\mathcal{E}} \mid)$.

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?**

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**:

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

Feld C: Pfeilindizes, nach Zielknoten sortiert. Dazu ein Einstiegsfeld D je Zielknoten $(1, 1, 3, 6, 8, 9)$.

Die Vorgänger von Knoten 3: Positionen 3 bis $5 \Rightarrow$ Pfeile $2, 5, 7$, also $(1, 3)$, $(4, 3)$ und $(5, 3)$.

Kürzeste Wege

Aufgabenstellung

Gesucht: die **kürzesten Wege** von einem Startknoten q 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.

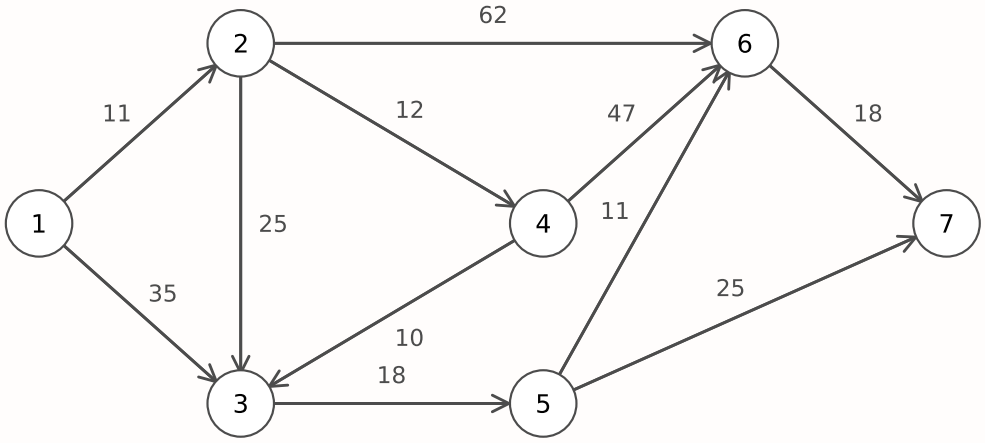
Ein Knoten steht **höchstens einmal** gleichzeitig in der Schlange, kann aber insgesamt **mehrfach** bearbeitet werden (Label-Correcting).

Iterative Vorgehensweise

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

- 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.

Beispiel: Busliniennetz



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

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

Der Ablauf im Detail

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

L = (1)

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

L = (~~1~~, 2, 3)

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

L = (~~1~~, ~~2~~, 3, 4, 6)

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

L = (~~1~~, ~~2~~, ~~3~~, 4, 6, 5)

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

L = (~~1~~, ~~2~~, ~~3~~, ~~4~~, 6, 5, 3)

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

L = (1, 2, 3, 4, 6, 5, 3, 7, 6, 5, 6, 7)

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

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

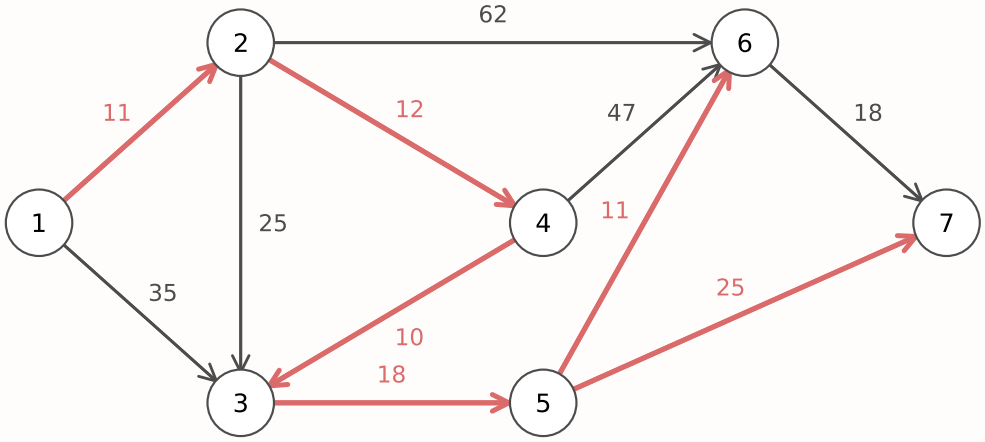
Knoten 2: 4 und 6 sind neu. Für 3 gilt $11 + 25 = 36 > 35$: **keine** Korrektur.

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

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

So läuft es weiter (5 korrigiert 6 und 7, das erneute 3 korrigiert 5, …), bis die Schlange **leer** ist.

Ergebnis als Baum



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

> ! Kürzeste Distanzen
>
> $$D = (0, 11, 33, 23, 51, 62, 76)$$

Varianten des Verfahrens

- **FIFO-LIFO**: ein erneut aufzunehmender Knoten wird **vorne** hinter den Schlangenkopf gesetzt, seine veralteten Nachfolger werden so zuerst korrigiert.
- **FIFO-FIFO**: mehrere erneut aufzunehmende Knoten behalten ihre Reihenfolge, eingefügt hinter dem zuletzt **eingefügten** Knoten.

Beide Varianten reduzieren **überflüssige** Aktualisierungen.

Netzwerkflussproblem

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).

Das Modell

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

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

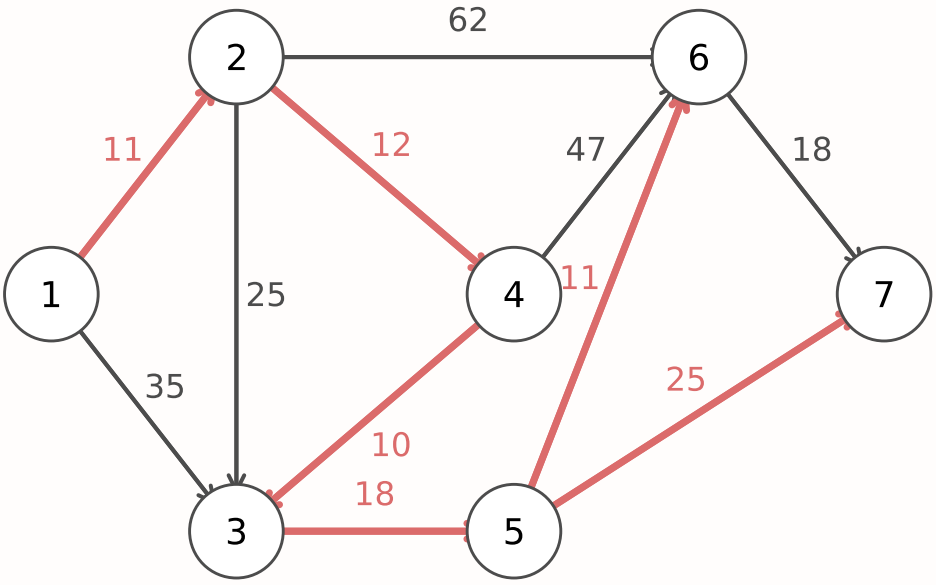
$$\sum_{j:(j,i)} X_{ji} - \sum_{j:(i,j)} X_{ij} = \begin{cases} 1 - |\,\mathcal{V}\,| & i = q \\ 1 & \text{sonst} \end{cases}$$

Kontrolle mit Julia

```
using JuMP, HiGHS

kanten = [(1,2,11),(1,3,35),(2,3,25),(2,4,12),(4,3,10),(3,5,18),
          (2,6,62),(4,6,47),(5,6,11),(5,7,25),(6,7,18)]
V, q = 1:7, 1

modell = Model(HiGHS.Optimizer); set_silent(modell)
@variable(modell, X[e in kanten] >= 0)                    # Fluss je Pfeil
@objective(modell, Min, sum(e[3] * X[e] for e in kanten))  # e = (i, j, Kosten)

for i in V                                                # Flusserhaltung
    zu = sum(X[e] for e in kanten if e[2] == i; init = 0.0)
    ab = sum(X[e] for e in kanten if e[1] == i; init = 0.0)
    @constraint(modell, zu - ab == (i == q ? 1 - length(V) : 1))
end

optimize!(modell)
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**:

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

! Schöner Zusammenhang

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

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

Zusammenfassung

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).
- Kürzeste Wege sind ein Spezialfall des **Netzwerkflussproblems**, lösbar als LP.

Literatur

Literaturverzeichnis

Bibliografie

[1] T. Vlćek, K. Haase, M. Fliedner, und T. Cors, „Police service district planning“, *OR Spectrum*, S. 1–33, 2024.