5.6 Kürzeste Wege (shortest paths)

5.6.1 Definitionen

- Gegeben sei ein (gerichteter oder ungerichteter) gewichteter Graph $G = (V, E, \rho)$.

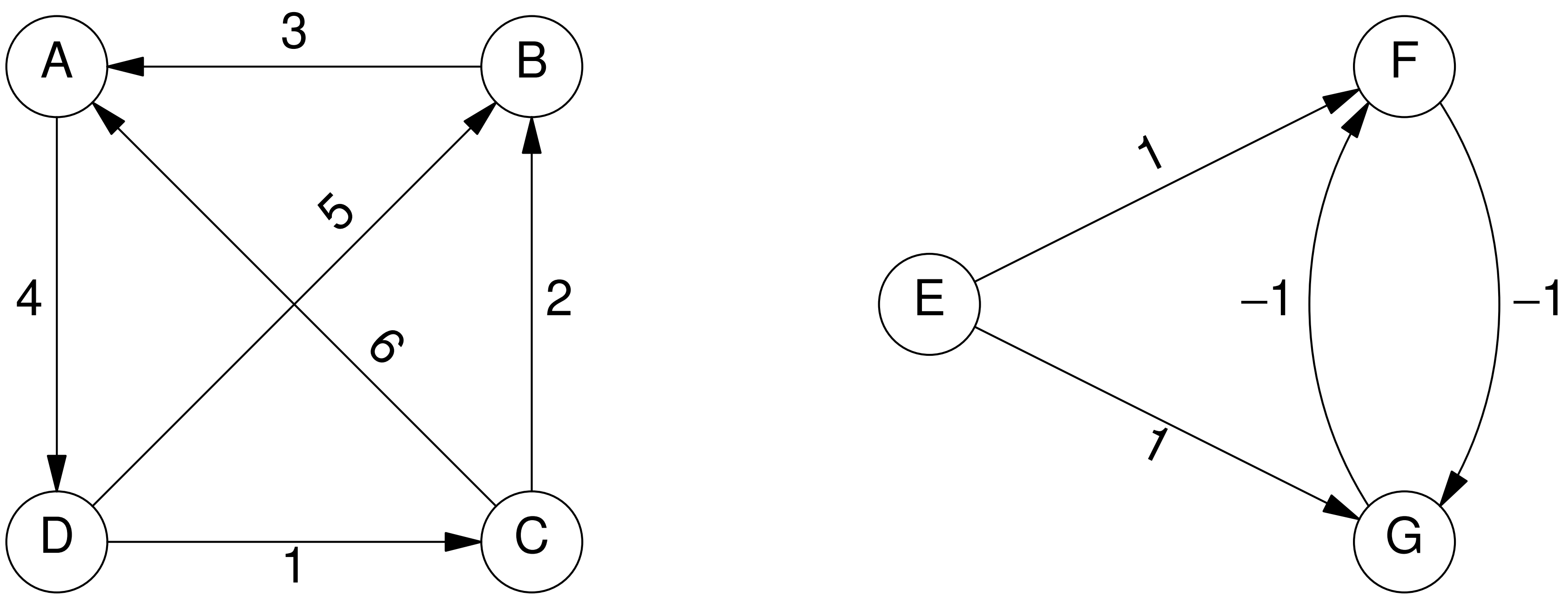
- Das *Gewicht* eines Wegs $w = w_0, \ldots, w_n$ ist die Summe $\rho(w) = \sum_{i=1}^{n} \rho(w_{i-1}, w_i)$ der Gewichte aller Kanten auf dem Weg.

- Wenn es einen Weg von einem Knoten u zu einem Knoten v gibt, ist ein *kürzester Weg* von u nach v ein Weg von u nach v mit minimalem Gewicht.
  (Weil ein Weg keinen Knoten mehrfach enthalten kann, kann es nur endlich viele verschiedene Wege von u nach v geben, sodass das Minimum wohldefiniert ist. Wenn es unendlich viele verschiedene Wege geben könnte, gäbe es u. U. keinen mit minimalem Gewicht.)

- In diesem Fall ist der *Abstand* $\delta(u, v)$ von u nach v das Gewicht eines kürzesten Wegs von u nach v. Andernfalls ist $\delta(u, v) = \infty$.

- Anmerkung: Da Kanten und Wege ein „Gewicht“ besitzen, müsste man eigentlich von „leichtesten“ statt von „kürzesten“ Wegen sprechen. Auch der Begriff des „Abstands“ passt eigentlich nicht zu dem des „Gewichts“. Trotzdem werden die Bezeichnungen üblicherweise so verwendet.

Beispiel



- In diesem Graphen gilt zum Beispiel:

  $\delta(A, B) = 7$

  $\delta(C, A) = 5$

  $\delta(A, E) = \delta(E, A) = \infty$

  $\delta(E, F) = \delta(E, G) = 0$

  $\delta(F, G) = \delta(G, F) = -1$

5.6.2 Problemstellungen

- Ein Knotenpaar:
  Finde einen kürzesten Weg von einem bestimmten Startknoten s zu einem bestimmten Zielknoten t.

- Fester Startknoten:
  Finde jeweils einen kürzesten Weg von einem bestimmten Startknoten s zu allen Knoten des Graphen.

- Fester Zielknoten:
  Finde jeweils einen kürzesten Weg von allen Knoten des Graphen zu einem bestimmten Zielknoten t.

- Alle Knotenpaare:
  Finde jeweils einen kürzesten Weg von jedem Knoten des Graphen zu jedem anderen.

Anmerkungen

- Das Problem mit festem Zielknoten lässt sich auf das Problem mit festem Startknoten zurückführen, indem man den transponierten Graphen betrachtet.

- Es sind keine Algorithmen bekannt, die das Problem für ein einzelnes Knotenpaar effizienter lösen als das Problem mit festem Startknoten.

- Das Problem für alle Knotenpaare lässt sich prinzipiell auf das Problem mit festem Startknoten zurückführen, indem man jeden Knoten des Graphen einmal als Startknoten wählt.
  Hier gibt es jedoch spezielle Algorithmen, die das Problem für alle Knotenpaare effizienter lösen.

- Aufgrund dieser Beobachtungen werden im folgenden nur Algorithmen für das Problem mit festem Startknoten sowie solche für alle Knotenpaare betrachtet.

5.6.3 Hilfsmittel für das Problem mit festem Startknoten

- Für jeden Knoten $v \in V$ wird das Gewicht $\delta(v)$ des kürzesten bis jetzt gefundenen Wegs vom Startknoten s zu v sowie der Vorgänger $\pi(v)$ von v auf diesem Weg gespeichert.

- Solange noch kein Weg von s nach v gefunden wurde, ist $\delta(v) = \infty$ und $\pi(v) = \perp$.

- Initialisierung:
  - Für alle Knoten $v \in V$: Setze $\delta(v) = \infty$ und $\pi(v) = \perp$.
  - Setze dann $\delta(s) = 0$.

- Verwerten einer Kante von u nach v:

  Wenn $\delta(u) + \rho(u, v) < \delta(v)$ ist,
  d. h. wenn der Weg von s nach v über u kürzer als der kürzeste bis jetzt gefundene Weg von s nach v ist:

  Setze $\delta(v) = \delta(u) + \rho(u, v)$ und $\pi(v) = u$,
  d. h. speichere den Weg über u als neuen kürzesten Weg nach v.

5.6.4 Algorithmus von Bellman und Ford

Gegeben

- Gewichteter Graph $G = (V, E, \rho)$, Startknoten $s \in V$
  (Das heißt, der Algorithmus löst das Problem mit festem Startknoten.)

- Einschränkung:

  Der Graph darf keine *negativen Zyklen* enthalten (d. h. Zyklen $w_0, \ldots, w_n$ mit $w_0 = w_n$, deren Gewicht $\sum_{i=1}^{n} \rho(w_{i-1}, w_i)$ negativ ist), die vom Startknoten s aus erreichbar sind.

- Die Einhaltung dieser Einschränkung wird vom Algorithmus selbst überprüft.
  Wenn sie verletzt ist, bricht der Algorithmus ab.

- Abgesehen davon, sind Kanten mit negativem Gewicht erlaubt.

Algorithmus

1 Für alle Knoten $v \in V$: Setze $\delta(v) = \infty$ und $\pi(v) = \perp$.
Setze dann $\delta(s) = 0$.

2 Wiederhole $(|V| - 1)$-mal:
Für jede Kante $(u, v) \in E$:
Verwerte die Kante (vgl. § 5.6.3).

3 Für jede Kante $(u, v) \in E$:
Wenn $\delta(u) + \rho(u, v) < \delta(v)$:
Abbruch, weil der Graph einen von s aus erreichbaren negativen Zyklus enthält.

Ergebnis

- Wenn der Graph einen von s aus erreichbaren negativen Zyklus enthält, bricht der Algorithmus in Schritt 3 ab.

- Andernfalls gilt nach Ausführung des Algorithmus für jeden Knoten $v \in V$:
  - $\delta(v) = \delta(s, v)$
  - Wenn $\pi(v) \neq \perp$ ist, ist $\pi(v)$ der Vorgänger von v auf einem kürzesten Weg von s nach v.

Laufzeit

- Initialisierung (Schritt 1): $O(|V|)$
- Eigentlicher Algorithmus (Schritt 2): $O(|V| \cdot |E|)$
- Überprüfung auf negative Zyklen (Schritt 3): $O(|E|)$
- Gesamtlaufzeit also: $O(|V| \cdot |E|)$

Beispiel

- Siehe § 5.6.1.

Korrektheit

Definition

- Eine *Route* (oder ein *Multiweg*) der Länge n von einem Knoten u zu einem Knoten v ist eine Folge von Knoten $w_0, \ldots, w_n$ mit $u = w_0$, Kanten von w_{i-1} nach w_i für $i = 1, \ldots, n$ und $w_n = v$.
  (Im Gegensatz zu einem Weg, darf ein Knoten in einer Route auch mehrmals vorkommen, d. h. eine Route darf auch Zyklen enthalten.)

Lemma 1

- Für jeden Knoten $v \in V$ gilt zu jedem Zeitpunkt entweder $\delta(v) = \infty$ oder $\delta(v) = \rho(w) = \sum_{i=1}^{n} \rho(w_{i-1}, w_i)$ für eine Route $w = w_0, \ldots, w_n$ vom Startknoten s zum Knoten v.

Beweis durch vollständige Induktion nach der Anzahl der bis jetzt verwerteten Kanten

- Am Anfang gilt für $v = s$: $\delta(v) = 0 = \rho(w)$ für die leere Route w von s nach v, und $\delta(v) = \infty$ für alle anderen Knoten $v \neq s$.

- Nach der Verwertung einer Kante (u, v) in Schritt 2 ist $\delta(v)$ entweder unverändert – dann folgt die Behauptung für diesen Knoten v direkt aus der Induktionsvoraussetzung –, oder es gilt: $\delta(v) = \delta(u) + \rho(u, v) = \rho(w) + \rho(u, v) = \rho(w')$ für eine Route $w = s, \ldots, u$ – die es nach Induktionsvoraussetzung gibt – und die Route $w' = s, \ldots, u, v$.
  Für alle anderen Knoten v bleibt $\delta(v)$ unverändert.

Lemma 2

- Nach dem n-ten Durchlauf der Schleife in Schritt 2 gilt für jede Route $w = s, \ldots, v$ der Länge n vom Startknoten s zu irgendeinem Knoten v: $\delta(v) \leq \rho(w)$.

Beweis durch vollständige Induktion nach n

- Induktionsanfang $n = 0$
  - Die einzige Route w der Länge 0 vom Startknoten s zu irgendeinem Knoten v ist die leere Route von s nach s, für die gilt: $\rho(w) = 0$.
  - Also gilt für $v = s$: $\delta(v) = 0 \leq \rho(w)$.
- Induktionsschritt $n \to n + 1$
  - Sei $w = s, \ldots, u, v$ eine Route der Länge $n + 1$ und $w' = s, \ldots, u$.
  - Nach dem n-ten Durchlauf der Schleife gilt nach Induktionsvoraussetzung: $\delta(u) \leq \rho(w')$.
  - Im $(n + 1)$-ten Durchlauf der Schleife wird u. a. die Kante (u, v) verwertet, woraus $\delta(v) \leq \delta(u) + \rho(u, v) \leq \rho(w') + \rho(u, v) = \rho(w)$ folgt.
  - Durch die Verwertung weiterer Kanten im selben oder späteren Durchläufen wird der Wert von $\delta(v)$ eventuell noch kleiner, aber niemals größer.

Lemma 3

- Wenn G keinen vom Startknoten s aus erreichbaren negativen Zyklus enthält, gilt für jeden Knoten $v \in V$ zu jedem Zeitpunkt: $\delta(v) \geq \delta(s, v)$.

Beweis

- Für $\delta(v) = \infty$ gilt trivialerweise $\delta(v) \geq \delta(s, v)$, unabhängig vom Wert von $\delta(s, v)$.
- Für $\delta(v) < \infty$ gilt nach Lemma 1: $\delta(v) = \rho(w)$ für eine Route w von s nach v.
- Da es keinen von s aus erreichbaren negativen Zyklus gibt, kann diese Route keinen negativen Zyklus enthalten.
- Für den Weg w' von s nach v, der entsteht, wenn man aus der Route w eventuell vorhandene Zyklen entfernt, gilt nach Definition von $\delta(s, v)$: $\rho(w') \geq \delta(s, v)$.
- Da das Gewicht der eventuell entfernten Zyklen nicht negativ ist, gilt: $\delta(v) = \rho(w) \geq \rho(w') \geq \delta(s, v)$.

Satz 1

- Wenn G keinen vom Startknoten s aus erreichbaren negativen Zyklus enthält, gilt nach Ausführung von Schritt 2 für jeden Knoten $v \in V$: $\delta(v) = \delta(s, v)$.

Beweis:

- Nach Lemma 3 gilt für jeden Knoten $v \in V$: $\delta(v) \geq \delta(s, v)$.
  Deshalb genügt es zu zeigen, dass für jeden Knoten $v \in V$ gilt: $\delta(v) \leq \delta(s, v)$.

- Wenn v von s aus nicht erreichbar ist, ist $\delta(s, v) = \infty$ und somit trivialerweise $\delta(v) \leq \delta(s, v)$.

- Wenn v von s aus erreichbar ist, gibt es einen Weg und damit auch einen kürzesten Weg $w = s, \ldots, v$ von s nach v, d. h. $\delta(s, v) = \rho(w)$.

- Da ein Weg keinen Knoten mehrmals enthalten kann, ist die Länge dieses Wegs höchstens $|V| - 1$.

- Deshalb gilt nach dem $(|V| - 1)$-ten Durchlauf der Schleife gemäß Lemma 2:
  $\delta(v) \leq \rho(w) = \delta(s, v)$.

Korollar

- Wenn G keinen vom Startknoten s aus erreichbaren negativen Zyklus enthält, bricht der Algorithmus in Schritt 3 nicht ab.

Beweis

- Annahme: Es gibt eine Kante $(u, v) \in E$, für die in Schritt 3 die Abbruchbedingung $\delta(u) + \rho(u, v) < \delta(v)$ erfüllt ist.
- Nach Satz 1 gilt zu diesem Zeitpunkt: $\delta(u) = \delta(s, u)$ und $\delta(v) = \delta(s, v)$.
  Also muss für die Kante (u, v) die Bedingung $\delta(s, u) + \rho(u, v) < \delta(s, v)$ erfüllt sein, woraus insbesondere $\delta(s, u) < \infty$ folgt.
- Deshalb gibt es einen kürzesten Weg $w = s, \ldots, u$ von s nach u mit $\rho(w) = \delta(s, u)$.
- Dann ist $w' = s, \ldots, u, v$ ein Weg von s nach v mit Gewicht $\delta(s, u) + \rho(u, v) < \delta(s, v)$.
- Widerspruch zur Definition von $\delta(s, v)$!
- Also gibt es keine Kante, für die in Schritt 3 die Abbruchbedingung erfüllt ist.

Satz 2

- Wenn G einen von s aus erreichbaren negativen Zyklus $w = w_0, \ldots, w_n$ enthält, bricht der Algorithmus in Schritt 3 ab.

Beweis

- Da $w_0, \ldots, w_n$ ein Zyklus ist, gilt $w_0 = w_n$ und deshalb $S = \sum_{i=1}^{n} \delta(w_i) = \sum_{i=1}^{n} \delta(w_{i-1})$.

- Da der Zyklus von s aus erreichbar ist, gibt es für jeden seiner Knoten w_i einen Weg w von s nach w_i mit Länge $\leq |V| - 1$.
  Deshalb gilt nach Lemma 2: $\delta(w_i) \leq \rho(w) < \infty$ für jeden Knoten w_i des Zyklus und somit auch $S = \sum_{i=1}^{n} \delta(w_i) < \infty$.

- Annahme: In Schritt 3 gilt für keine Kante $(u, v) \in E$ die Abbruchbedingung $\delta(u) + \rho(u, v) < \delta(v)$, d. h. für jede Kante $(u, v) \in E$ gilt: $\delta(v) \leq \delta(u) + \rho(u, v)$.

- Insbesondere gilt $\delta(w_i) \leq \delta(w_{i-1}) + \rho(w_{i-1}, w_i)$ für jede Kante (w_{i-1}, w_i) des Zyklus und somit: $S = \sum_{i=1}^{n} \delta(w_i) \leq \sum_{i=1}^{n} (\delta(w_{i-1}) + \rho(w_{i-1}, w_i)) = \sum_{i=1}^{n} \delta(w_{i-1}) + \sum_{i=1}^{n} \rho(w_{i-1}, w_i) = S + \rho(w)$.

- Daraus folgt wegen $S < \infty$: $0 \leq \rho(w)$. Widerspruch, da w ein negativer Zyklus ist.

5.6.5 Algorithmus von Dijkstra

Gegeben

- Gewichteter Graph $G = (V, E, \rho)$, Startknoten $s \in V$
  (Das heißt, der Algorithmus löst das Problem mit festem Startknoten.)

- Einschränkung:
  Der Graph darf keine Kanten mit negativem Gewicht enthalten.

- Die Einhaltung dieser Einschränkung wird vom Algorithmus *nicht* überprüft.
  Wenn sie verletzt ist, ist das Ergebnis des Algorithmus undefiniert.

Algorithmus

1 Für alle Knoten $v \in V$: Setze $\delta(v) = \infty$ und $\pi(v) = \perp$.
Setze dann $\delta(s) = 0$.

2 Für alle Knoten $v \in V$:
Füge v mit Priorität $\delta(v)$ in eine Minimum-Vorrangwarteschlange ein.

3 Solange die Warteschlange nicht leer ist:

   1 Entnimm einen Knoten u mit minimaler Priorität.

   2 Für jeden Nachfolger v von u, der sich noch in der Warteschlange befindet:

      1 Verwerte die Kante (u, v) (vgl. § 5.6.3).

      2 Wenn $\delta(v)$ dadurch erniedrigt wurde:

      Erniedrige die Priorität von v in der Warteschlange entsprechend.

Ergebnis

- Wenn der Graph keine Kanten mit negativem Gewicht enthält, gilt nach Ausführung des Algorithmus für alle Knoten $v \in V$:
  - $\delta(v) = \delta(s, v)$
  - Wenn $\pi(v) \neq \perp$ ist, ist $\pi(v)$ der Vorgänger von v auf einem kürzesten Weg von s nach v.
- Andernfalls ist das Ergebnis des Algorithmus undefiniert.

Beispiel

- Siehe § 5.6.1.

Laufzeit

- Initialisierung (Schritt 1): $O(|V|)$

- Operationen auf der Vorrangwarteschlange:
  - $|V|$-mal Einfügen eines Knotens
  - $|V|$-mal Test, ob die Warteschlange leer ist
  - $|V|$-mal Entnehmen eines Knotens mit minimaler Priorität
  - $|E|$-mal Test, ob ein Knoten enthalten ist
  - Maximal $|E|$-mal Erniedrigen der Priorität eines Knotens
    (Der Rumpf der inneren Schleife wird höchstens $|E|$-mal ausgeführt.)

  Insgesamt $O(|V| + |E|)$

- Laufzeit jeder solchen Operation: $O(\log |V|)$,
  da die Warteschlange maximal $|V|$ Einträge enthält.

- Gesamtlaufzeit somit: $O((|V| + |E|) \log |V|)$

Korrektheit

1. Sei Q die Menge der Knoten, die sich zu einem bestimmten Zeitpunkt noch in der Warteschlange befinden, und $P = V \setminus Q$.

2. Am Ende jedes Durchlaufs durch die Schleife in Schritt 3 gilt: $\delta(v) \le \delta(u) + \rho(u, v)$ für jeden Nachfolger v von u, der sich noch in der Warteschlange befindet.
   (Dies folgt unmittelbar aus der Definition der Operation „Verwerten".)

3. Zum Zeitpunkt der Entnahme eines Knotens u aus der Warteschlange gilt: $\delta(u) = \delta(s, u)$.

4. Da $\delta(u)$ für Knoten $u \in P$ nicht mehr verändert wird und $\delta(v)$ für Knoten $v \in Q$ später höchstens noch verkleinert wird, gelten diese Aussagen auch zu jedem späteren Zeitpunkt.

5. Damit gilt insbesondere nach Ausführung des Algorithmus: $\delta(u) = \delta(s, u)$ für jeden Knoten $u \in V$.

Beweis von Aussage 3 durch „Schleifeninduktion":

❐ Zum Zeitpunkt der Entnahme von $u = s$ gilt: $\delta(s) = 0 = \delta(s, s)$

❐ Zum Zeitpunkt der Entnahme eines anderen Knotens u gilt:

- Da es keine negativen Kantengewichte und somit auch keine negativen Zyklen gibt, gilt nach Lemma 3 in § 5.6.4: $\delta(u) \geq \delta(s, u)$.

- Wenn u von s aus nicht erreichbar ist, gilt somit: $\delta(u) \geq \delta(s, u) = \infty \Rightarrow \delta(u) = \delta(s, u)$.

- Wenn u von s aus erreichbar ist, gibt es einen kürzesten Weg $w = s, \ldots, u$ von s nach u mit Gewicht $\rho(w) = \delta(s, u)$.

- Sei p der letzte Knoten auf diesem Weg, für den $p \in P$ gilt (eventuell ist $p = s$), und q der nächste Knoten auf diesem Weg (eventuell ist $q = u$), d. h. q ist ein Nachfolger von p, und es gilt $q \in Q$.

- Somit gilt: $\delta(s, u) = \rho(w) = \rho(s, \ldots, p, q, \ldots, u) = \rho(s, \ldots, p) + \rho(p, q) + \rho(q, \ldots, u) \underset{(a)}{\geq}$

  $\rho(s, \ldots, p) + \rho(p, q) \underset{(b)}{\geq} \delta(s, p) + \rho(p, q) \underset{(c)}{=} \delta(p) + \rho(p, q) \underset{(d)}{\geq} \delta(q) \underset{(e)}{\geq} \delta(u)$, denn:

  a) $\rho(q, \ldots, u) \geq 0$, da es keine negativen Kantengewichte gibt.

  b) $\rho(s, \ldots, p) \geq \delta(s, p)$ für jeden Weg von s nach p

  c) Induktionsvoraussetzung

  d) Aussage 2

  e) u ist der Knoten mit minimaler Priorität, d. h. $\delta(u) \leq \delta(q)$ für alle $q \in Q$.

- Damit gilt einerseits $\delta(u) \geq \delta(s, u)$ und andererseits $\delta(u) \leq \delta(s, u)$ und somit $\delta(u) = \delta(s, u)$.

5.6.6 Algorithmus von Floyd und Warshall

Gegeben

- Gewichteter Graph $G = (V, E, \rho)$
  (Das heißt, der Algorithmus löst das Problem für alle Knotenpaare.)

- Einschränkung: Der Graph darf keine negativen Zyklen enthalten.

- Die Einhaltung dieser Einschränkung wird vom Algorithmus *nicht* überprüft.
  Wenn sie verletzt ist, ist das Ergebnis des Algorithmus undefiniert.

- Abgesehen davon, sind Kanten mit negativem Gewicht erlaubt.

Definitionen

- Zur Vereinfachung der Notation sei die Knotenmenge des Graphen $V = \{1, \ldots, m\}$.

- Für $k = 0, \ldots, m$ sei ein Weg von u nach v *via* k ein Weg $w_0, \ldots, w_n$ mit $w_0 = u$, $w_n = v$ und $w_1, \ldots, w_{n-1} \leq k$, d. h. alle *Zwischenknoten* (sofern es welche gibt) gehören zur Menge $\{1, \ldots, k\}$ (die für $k = 0$ leer ist).

- Für $k = 0, \ldots, m$ und $u, v \in V$ sei
  - $\Delta_k(u, v)$ entweder das Gewicht eines kürzesten Wegs von u nach v via k oder ∞ (falls es keinen solchen Weg gibt)
  - $\Pi_k(u, v)$ entweder der Vorgänger von v auf diesem Weg oder $\perp$.

Rekursionsgleichungen

- Für $k = 0$ gilt:

$\Delta_0(u, v) =$	$\Pi_0(u, v) =$	wenn
0	$\perp$	$u = v$
$\rho(u, v)$	u	$u \neq v$ und $(u, v) \in E$ bzw. $\{u, v\} \in E$
∞	$\perp$	sonst

- Für $k = 1, \ldots, m$ gilt:

$\Delta_k(u, v) =$	$\Pi_k(u, v) =$	wenn
$\Delta_{k-1}(u, v)$	$\Pi_{k-1}(u, v)$	$\Delta_{k-1}(u, v) \leq \Delta_{k-1}(u, k) + \Delta_{k-1}(k, v)$
$\Delta_{k-1}(u, k) + \Delta_{k-1}(k, v)$	$\Pi_{k-1}(k, v)$	$\Delta_{k-1}(u, v) > \Delta_{k-1}(u, k) + \Delta_{k-1}(k, v)$

Begründung

Mit den Bezeichnungen $d = \Delta_k(u, v)$, $d_1 = \Delta_{k-1}(u, v)$ und $d_2 = \Delta_{k-1}(u, k) + \Delta_{k-1}(k, v)$ gelten folgende Aussagen:

1. $d = d_1$ oder $d = d_2$

   Beweis:

   ❍ Wenn es keinen Weg von u nach v via k gibt, ist $d = \infty$.
   In diesem Fall gibt es auch keinen Weg von u nach v via $k - 1$, d. h. es ist auch $d_1 = \infty$ und somit $d = d_1$.

   ❍ Wenn es einen kürzesten Weg von u nach v via k (mit Gewicht d) gibt und dieser den Knoten k *nicht* enthält, stimmt er mit einem kürzesten Weg von u nach v via $k - 1$ (mit Gewicht d_1) überein, d. h. es gilt wiederum $d = d_1$.

   ❍ Andernfalls besteht dieser Weg (mit Gewicht d) aus einem (eventuell leeren) kürzesten Teilweg von u nach k via $k - 1$ (mit Gewicht $\Delta_{k-1}(u, k)$) und einem (eventuell leeren) kürzesten Teilweg von k nach v via $k - 1$ (mit Gewicht $\Delta_{k-1}(k, v)$), d. h. es gilt $d = d_2$.
   (Beachte: Jeder Teilweg eines kürzesten Wegs ist ebenfalls ein kürzester Weg, sofern es keine negativen Zyklen gibt.)

2. $d = \min(d_1, d_2)$

Anmerkung:

- Diese Aussage folgt *nicht* unmittelbar aus der vorigen Aussage 1!
- Wenn ein Graph einen negativen Zyklus enthält, kann $d = d_1$ gelten, obwohl $d_1 > d_2$ ist.
- Deshalb verwendet der folgende Beweis an einer entscheidenden Stelle die Voraussetzung, dass der Graph *keinen* negativen Zyklus enthält.

Beweis:

- Wenn $d_1 = d_2$ ist (insbesondere auch, wenn $d_1 = d_2 = \infty$ ist), folgt die Behauptung unmittelbar aus der vorigen Aussage 1.
- Wenn $d_1 < d_2$ ist (woraus insbesondere $d_1 < \infty$ folgt), dann gibt es einen kürzesten Weg von u nach v via $k - 1$ mit Gewicht d_1, der auch ein Weg von u nach v via k ist.
  Daraus folgt $d \le d_1$, und zusammen mit $d_1 < d_2$ und Aussage 1:
  $d = d_1 = \min(d_1, d_2)$.

- Wenn $d_2 < d_1$ ist (woraus insbesondere $d_2 < \infty$ folgt), dann gibt es einen kürzesten Weg von u nach k via $k - 1$ mit Gewicht $\Delta_{k-1}(u, k)$ sowie einen kürzesten Weg von k nach v via $k - 1$ mit Gewicht $\Delta_{k-1}(k, v)$.
  - Wenn man diese Wege zusammensetzt, entsteht eine *Route* von u nach v via k mit Gewicht d_2.
  - Annahme: Diese Route $u, \ldots, k, \ldots, v$ ist kein Weg, d. h. sie enthält (mindestens) einen Zyklus $w, \ldots, w$.
  - Da die Teilrouten $u, \ldots, k$ und $k, \ldots, v$ Wege sind, d. h. keinen Knoten mehrfach enthalten, muss der Zyklus den Knoten k enthalten, d. h. die Route muss $u, \ldots, w, \ldots, k, \ldots, w, \ldots, v$ lauten.
  - Wenn man den Zyklus (bzw. die Zyklen) entfernt, entsteht somit ein Weg von u nach v via $k - 1$ mit Gewicht $\leq d_2$, da das Gewicht der Zyklen nicht negativ ist.
  - Zusammen mit $d_2 < d_1$ ist dies ein Widerspruch dazu, dass d_1 das Gewicht eines kürzesten Wegs von u nach v via $k - 1$ ist.
  - Also muss die o. g. Route immer ein *Weg* von u nach v via k mit Gewicht d_2 sein.
  - Daraus folgt $d \leq d_2$, und zusammen mit $d_2 < d_1$ und Aussage 1: $d = d_2 = \min(d_1, d_2)$.

Praktische Berechnung

- Zur Berechnung des Werts $\Delta_k(u, v)$ in Zeile u und Spalte v der Matrix Δ_k werden neben dem korrespondierenden Wert $\Delta_{k-1}(u, v)$ der „vorigen" Matrix Δ_{k-1} noch die Werte $\Delta_{k-1}(u, k)$ und $\Delta_{k-1}(k, v)$ benötigt, die sich in Spalte bzw. Zeile k dieser Matrix befinden.

- Diese Werte in Spalte und Zeile k bleiben beim Übergang von Δ_{k-1} zu Δ_k unverändert, denn es gilt:

  - Für $v = k$ (d. h. Spalte k):
    $\Delta_k(u, v) = \min(\Delta_{k-1}(u, v), \Delta_{k-1}(u, k) + \Delta_{k-1}(k, v)) =$
    $= \min(\Delta_{k-1}(u, k), \Delta_{k-1}(u, k) + \Delta_{k-1}(k, k)) =$
    $= \min(\Delta_{k-1}(u, k), \Delta_{k-1}(u, k) + 0) = \Delta_{k-1}(u, k) = \Delta_{k-1}(u, v)$

  - Für $u = k$ (d. h. Zeile k):
    $\Delta_k(u, v) = \min(\Delta_{k-1}(u, v), \Delta_{k-1}(u, k) + \Delta_{k-1}(k, v)) =$
    $= \min(\Delta_{k-1}(k, v), \Delta_{k-1}(k, k) + \Delta_{k-1}(k, v)) =$
    $= \min(\Delta_{k-1}(k, v), 0 + \Delta_{k-1}(k, v)) = \Delta_{k-1}(k, v) = \Delta_{k-1}(u, v)$

  (Beachte: $\Delta_{k-1}(k, k) = 0$, weil ein kürzester Weg von k nach k immer Gewicht 0 besitzt.)

- Somit können die Werte der Matrix Δ_{k-1} (und analog Π_{k-1}) jeweils gefahrlos durch die korrespondierenden Werte der Matrix Δ_k (und analog Π_k) überschrieben werden.

Algorithmus

1 Für $u = 1, \ldots, m$:

  1 Für $v = 1, \ldots, m$: Setze $\Delta(u, v) = \infty$ und $\Pi(u, v) = \perp$.

  2 Für jeden Nachfolger v von u: Setze $\Delta(u, v) = \rho(u, v)$ und $\Pi(u, v) = u$.

  3 Setze $\Delta(u, u) = 0$ und $\Pi(u, u) = \perp$.

2 Für $k = 1, \ldots, m$:

  Für $u = 1, \ldots, m$ und $v = 1, \ldots, m$:

  Wenn $\Delta(u, v) > \Delta(u, k) + \Delta(k, v)$:

  Setze $\Delta(u, v) = \Delta(u, k) + \Delta(k, v)$ und $\Pi(u, v) = \Pi(k, v)$.

Ergebnis

- Nach Ausführung des Algorithmus gilt für alle Knoten $u, v \in V$:
  - $\Delta(u, v) = \Delta_m(u, v)$ = Gewicht eines kürzesten Wegs von u nach v via m = $\delta(u, v)$ = Gewicht eines beliebigen kürzesten Wegs von u nach v, falls ein solcher Weg existiert, andernfalls ∞
  - $\Pi(u, v) = \Pi_m(u, v)$ = Vorgänger von v auf diesem Weg bzw. $\perp$

Laufzeit

❐ Offensichtlich $O(m^3) = O(|V|^3)$

Beispiel

❐ Siehe § 5.6.1.

5.6.7 Laufzeitvergleich

Fester Startknoten

- Das Problem mit festem Startknoten kann mit folgenden Algorithmen gelöst werden, sofern die jeweilige Voraussetzung erfüllt ist:
  - Dijkstra, wenn es keine negativen Kantengewichte gibt
  - Bellman-Ford,
    wenn es keinen vom Startknoten aus erreichbaren negativen Zyklus gibt
  - Floyd-Warshall, wenn es überhaupt keinen negativen Zyklus gibt
- Die nachfolgende Tabelle zeigt die jeweiligen Laufzeiten im Vergleich

Algorithmus	*Laufzeit*		
	allgemein	wenn $\lvert E\rvert \approx \lvert V\rvert$	wenn $\lvert E\rvert \approx \lvert V\rvert^2$
Dijkstra	$O((\lvert V\rvert + \lvert E\rvert) \log \lvert V\rvert)$	$O(\lvert V\rvert \log \lvert V\rvert)$	$O\left(\lvert V\rvert^2 \log \lvert V\rvert\right)$
Bellman-Ford	$O(\lvert V\rvert \cdot \lvert E\rvert)$	$O\left(\lvert V\rvert^2\right)$	$O\left(\lvert V\rvert^3\right)$
Floyd-Warshall	$O\left(\lvert V\rvert^3\right)$	$O\left(\lvert V\rvert^3\right)$	$O\left(\lvert V\rvert^3\right)$

Alle Knotenpaare

- Das Problem für alle Knotenpaare kann mit folgenden Algorithmen gelöst werden, sofern die jeweilige Voraussetzung erfüllt ist:
  - $|V|$-mal Dijkstra, wenn es keine negativen Kantengewichte gibt
  - $|V|$-mal Bellman-Ford, wenn es keinen negativen Zyklus gibt
  - 1-mal Floyd-Warshall, wenn es keinen negativen Zyklus gibt
- Die nachfolgende Tabelle zeigt die jeweiligen Gesamtlaufzeiten im Vergleich

Algorithmus	*Laufzeit*		
	allgemein	wenn $|E| \approx |V|$	wenn $|E| \approx |V|^2$
Dijkstra	$O(|V|\,(|V| + |E|)\log|V|)$	$O\left(|V|^2 \log|V|\right)$	$O\left(|V|^3 \log|V|\right)$
Bellman-Ford	$O\left(|V|^2 \cdot |E|\right)$	$O\left(|V|^3\right)$	$O\left(|V|^4\right)$
Floyd-Warshall	$O\left(|V|^3\right)$	$O\left(|V|^3\right)$	$O\left(|V|^3\right)$

5.7 Das Problem des Handlungsreisenden (Traveling Salesman Problem)

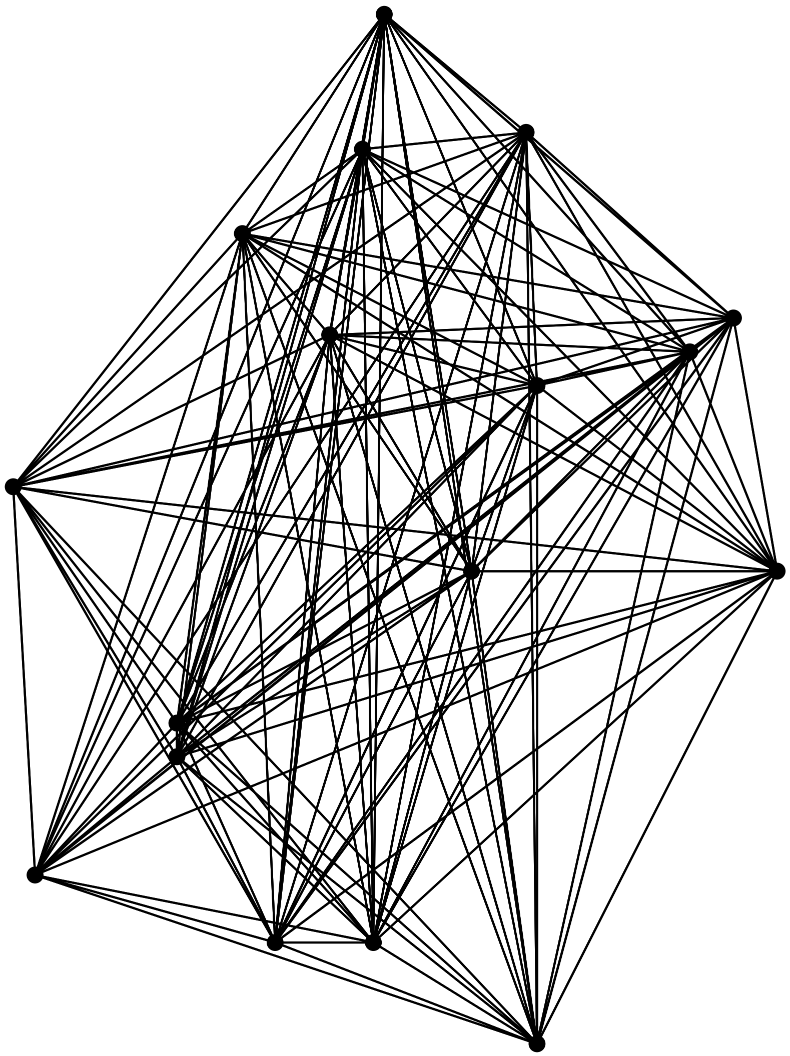
5.7.1 Problemstellung

Anschauliche Formulierung

- Ein Vertreter einer Aalener Firma betreut Kunden in vielen Städten Deutschlands.
- Um ihnen ein neues Produkt vorzustellen, muss er alle Kunden besuchen.
- Die Distanz zwischen je zwei Städten ist bekannt (z. B. Entfernungstabelle).
- Wie findet der Vertreter die kürzeste Tour, um von Aalen aus jede Stadt genau einmal zu besuchen und am Schluss wieder in Aalen zu sein (d. h. eine Rundtour)?

Mathematische Formulierung

Gegeben



- Ungerichteter, gewichteter Graph $G = (V, E, \rho)$
- mit $|V| = N$ Knoten,
- vollständiger Kantenmenge $E = \{ \{ u, v \} \mid u, v \in V \}$
- und Gewichtsfunktion $\rho : E \to \mathbb{R}^+$

Gesucht

- Zyklus $v_1, \ldots, v_N, v_1$, der jeden Knoten des Graphen genau einmal enthält,
- mit minimalem Gewicht,
- d. h. $\sum_{i=1}^{N-1} \rho(v_i, v_{i+1}) + \rho(v_N, v_1)$ ist minimal

Anwendungsmöglichkeiten

- Routenplanung
- Schaltungsentwurf/-verdrahtung
- Umrüsten von Produktionsmaschinen
- Usw.
- Musteranwendung und Benchmark für Approximationsverfahren

5.7.2 Exakte Lösungsverfahren

- Naiv: Alle Kombinationen ausprobieren: $O(N!)$
- Verbesserung durch sog. dynamisches Programmieren: $O\left(N^2 \cdot 2^N\right)$
- Weitere Verbesserungen z. B. durch Ausnutzen geometrischer Eigenschaften
- Trotzdem: Problem ist *NP-schwierig*,
  d. h. nach heutigem Wissenstand nicht effizient lösbar

5.7.3 Approximationsverfahren

Grundprinzip

- Finde effiziente Algorithmen,
  die eine möglichst gute *Näherungslösung* des Problems liefern

- Betrachte zwei Parameter zum Vergleich von Verfahren:
  - *Laufzeit* in Abhängigkeit von der Knotenzahl N
  - *Qualitätsfaktor* $q = \frac{\text{max. Gewicht der gefundenen Tour}}{\text{Gewicht einer optimalen Tour}}$

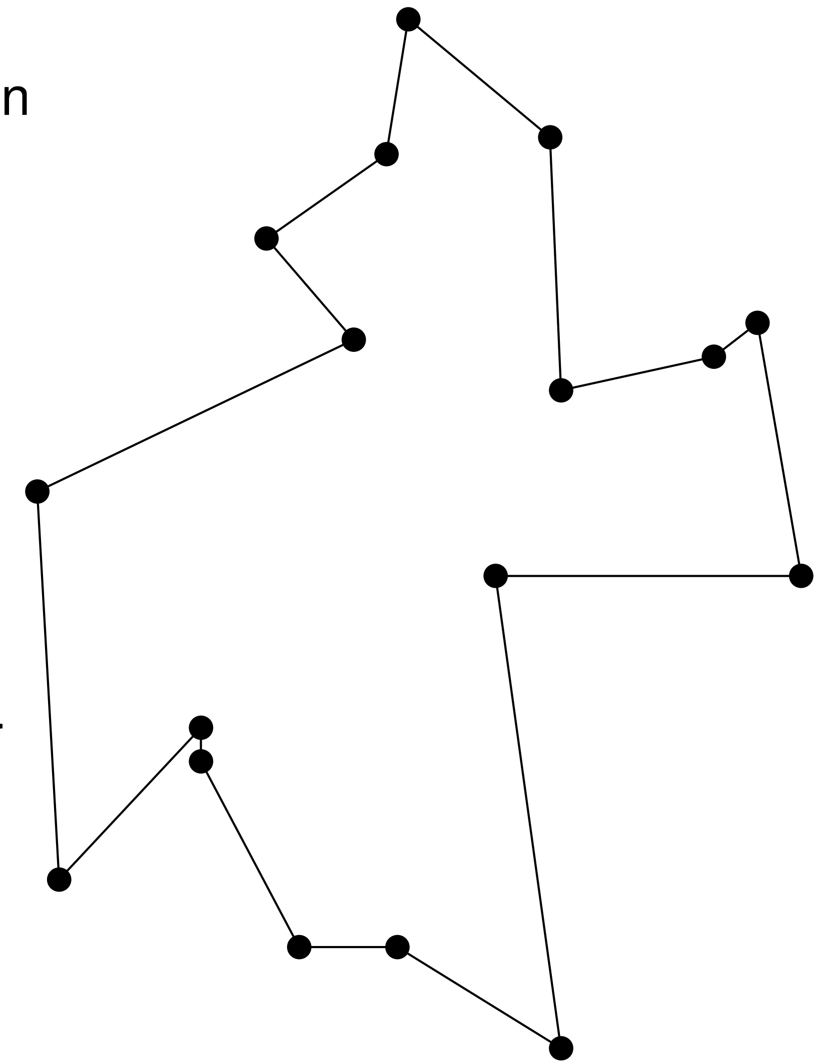
    ($q = 1$ bei exakten Verfahren)

- In der Regel wird die *Dreiecksungleichung* ausgenutzt:

  $$\rho(u, w) \leq \rho(u, v) + \rho(v, w) \qquad \text{für alle } u, v, w \in V$$

Das Nearest-Neighbour-Verfahren

- Beginne mit dem Startknoten v_1.
- Wähle als zweiten Knoten v_2 den nächstgelegenen Nachbarn von v_1 ($N - 1$ Kandidaten).
- Wähle als dritten Knoten v_3 den nächstgelegenen Nachbarn von v_2 aus der Menge der verbleibenden Knoten ($N - 2$ Kandidaten).
- Usw.
- Wähle als vorletzten Knoten v_{N-1} den nächstgelegenen Nachbarn von v_{N-2} aus der Menge der verbleibenden Knoten (2 Kandidaten).
- Füge den letzten verbleibenden Knoten v_N hinzu (1 Kandidat).
- Typisches Beispiel eines Nächstbest-Algorithmus



Laufzeit

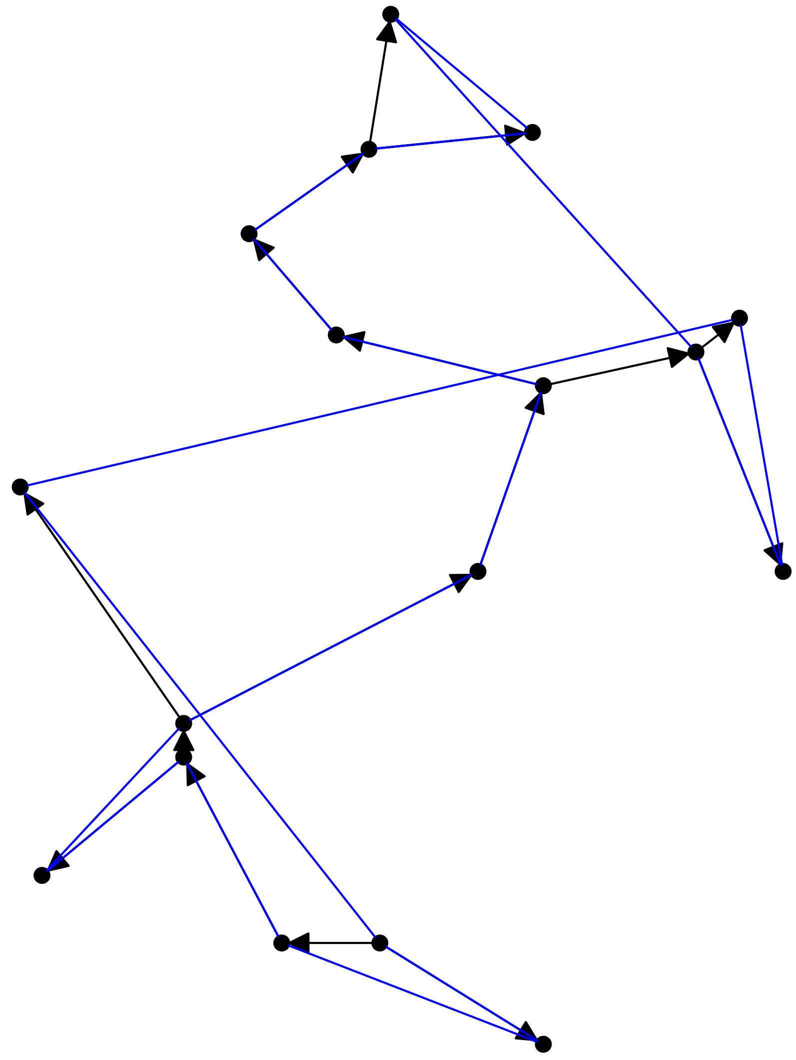
- $(N-1)+(N-2)+\ldots+2+1=\dfrac{(N-1)\,N}{2}=\mathrm{O}\left(N^2\right)$

Qualitätsfaktor

- Allgemein: $q(N)=\dfrac{\lfloor \log_2 N+1\rfloor}{2}$
- Konkret z. B.: $q(10)=2.5$ $\quad q(100)=4$ $\quad q(1000)=5.5$ $\quad q\left(10^6\right)=10.5$

Das Spannbaum-Verfahren

- Konstruiere einen minimalen Spannbaum des Graphen (schwarze Pfeile).
- Konstruiere eine Tour, die jede Kante des Spannbaums genau zweimal durchläuft, einmal „abwärts“ und einmal „aufwärts“.
- Überspringe alle Knoten, die bereits besucht wurden.



Laufzeit

- Konstruktion des minimalen Spannbaums mit dem Algorithmus von Prim:
  $\mathrm{O}(|E| \log |V|) = \mathrm{O}\left(N^2 \log N\right)$
- Durchlaufen des Spannbaums: $\mathrm{O}(N)$
- Insgesamt also: $\mathrm{O}\left(N^2 \log N\right)$

Qualitätsfaktor

- $L_{\mathrm{Tour}} \leq 2\, L_{\mathrm{MST}}$ (Dreiecksungleichung)
- $L_{\mathrm{MST}} \leq L_{\mathrm{ST}} \leq L_{\mathrm{Opt}}$ (jede Tour impliziert einen Spannbaum)
- Somit: $L_{\mathrm{Tour}} \leq 2\, L_{\mathrm{Opt}}$
- Also: $q = 2$