$G = (V, E)$ sei **gerichteter Graph** mit **Kantengewichten** $w : E \to \mathbb{R}$.

$G = (V, E)$ sei **gerichteter Graph** mit **Kantengewichten** $w : E \to \mathbb{R}$.

Definition 5.16

Es seien $s \in V$ und $t \in V$ zwei beliebige Knoten und es sei $P = (v_0, v_1, \ldots, v_\ell)$ ein Weg von s nach t. Wir definieren die **Länge von P** als $w(P) = \sum_{i=0}^{\ell-1} w(v_i, v_{i+1})$.

$G = (V, E)$ sei **gerichteter Graph** mit **Kantengewichten** $w : E \to \mathbb{R}$.

Definition 5.16

Es seien $s \in V$ und $t \in V$ zwei beliebige Knoten und es sei $P = (v_0, v_1, \ldots, v_\ell)$ ein Weg von s nach t. Wir definieren die **Länge von P** als $w(P) = \sum_{i=0}^{\ell-1} w(v_i, v_{i+1})$. Wir sagen, dass P ein **kürzester Weg von s nach t** ist, falls es keinen Weg P' von s nach t mit $w(P') < w(P)$ gibt.

$G = (V, E)$ sei **gerichteter Graph** mit **Kantengewichten** $w : E \to \mathbb{R}$.

Definition 5.16

Es seien $s \in V$ und $t \in V$ zwei beliebige Knoten und es sei $P = (v_0, v_1, \ldots, v_\ell)$ ein Weg von s nach t. Wir definieren die **Länge von P** als $w(P) = \sum_{i=0}^{\ell-1} w(v_i, v_{i+1})$. Wir sagen, dass P ein **kürzester Weg von s nach t** ist, falls es keinen Weg P' von s nach t mit $w(P') < w(P)$ gibt. Wir nennen die Länge $w(P)$ des kürzesten Weges P die **Entfernung von s nach t** und bezeichnen diese mit $\delta(s, t)$. Existiert kein s-t-Weg, so gelte $\delta(s, t) = \infty$.

Kürzeste-Wege-Probleme:

1. Im **Single-Source Shortest Path Problem (SSSP)** ist zusätzlich zu G und w ein Knoten $s \in V$ gegeben und wir möchten für jeden Knoten $v \in V$ einen kürzesten Weg von s nach v und die Entfernung $\delta(s, v)$ berechnen.

Kürzeste-Wege-Probleme:

1. Im **Single-Source Shortest Path Problem (SSSP)** ist zusätzlich zu G und w ein Knoten $s \in V$ gegeben und wir möchten für jeden Knoten $v \in V$ einen kürzesten Weg von s nach v und die Entfernung $\delta(s, v)$ berechnen.
2. Im **All-Pairs Shortest Path Problem (APSP)** sind nur G und w gegeben und wir möchten für jedes Paar $u, v \in V$ von Knoten einen kürzesten Weg von u nach v und die Entfernung $\delta(u, v)$ berechnen.

Kürzeste-Wege-Probleme:

1. Im **Single-Source Shortest Path Problem (SSSP)** ist zusätzlich zu G und w ein Knoten $s \in V$ gegeben und wir möchten für jeden Knoten $v \in V$ einen kürzesten Weg von s nach v und die Entfernung $\delta(s, v)$ berechnen.
2. Im **All-Pairs Shortest Path Problem (APSP)** sind nur G und w gegeben und wir möchten für jedes Paar $u, v \in V$ von Knoten einen kürzesten Weg von u nach v und die Entfernung $\delta(u, v)$ berechnen.

Für den Spezialfall, dass $w(e) = 1$ für alle Kanten $e \in E$ gilt, haben wir mit Breitensuche bereits einen Algorithmus kennengelernt, der das SSSP löst.

5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Lemma 5.17

Sei $P = (v_0, \ldots, v_\ell)$ ein kürzester Weg von $v_0 \in V$ nach $v_\ell \in V$. Für jedes Paar i, j mit $0 \leq i \leq j \leq \ell$ ist $P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ ein kürzester Weg von v_i nach v_j.

5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Lemma 5.17

Sei $P = (v_0, \ldots, v_\ell)$ ein kürzester Weg von $v_0 \in V$ nach $v_\ell \in V$. Für jedes Paar i, j mit $0 \leq i \leq j \leq \ell$ ist $P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ ein kürzester Weg von v_i nach v_j.

Beweis:

Zerlege P wie folgt:

$$P = v_0 \overset{P_{0i}}{\rightsquigarrow} v_i \overset{P_{ij}}{\rightsquigarrow} v_j \overset{P_{j\ell}}{\rightsquigarrow} v_\ell.$$

Dann gilt $w(P) = w(P_{0i}) + w(P_{ij}) + w(P_{j\ell})$.

5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Lemma 5.17

Sei $P = (v_0, \ldots, v_\ell)$ ein kürzester Weg von $v_0 \in V$ nach $v_\ell \in V$. Für jedes Paar i, j mit $0 \leq i \leq j \leq \ell$ ist $P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ ein kürzester Weg von v_i nach v_j.

Beweis:

Zerlege P wie folgt:

$$P = v_0 \overset{P_{0i}}{\rightsquigarrow} v_i \overset{P_{ij}}{\rightsquigarrow} v_j \overset{P_{j\ell}}{\rightsquigarrow} v_\ell.$$

Dann gilt $w(P) = w(P_{0i}) + w(P_{ij}) + w(P_{j\ell})$.

Ist P_{ij} kein kürzester v_i-v_j-Weg, so sei Weg P'_{ij} ein kürzerer v_i-v_j-Weg.
Mit diesem erhalten wir einen v_0-v_ℓ-Weg P' von v_0 nach v_ℓ:

$$P' = v_0 \overset{P_{0i}}{\rightsquigarrow} v_i \overset{P'_{ij}}{\rightsquigarrow} v_j \overset{P_{j\ell}}{\rightsquigarrow} v_\ell.$$

5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Lemma 5.17

Sei $P = (v_0, \ldots, v_\ell)$ ein kürzester Weg von $v_0 \in V$ nach $v_\ell \in V$. Für jedes Paar i, j mit $0 \leq i \leq j \leq \ell$ ist $P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ ein kürzester Weg von v_i nach v_j.

Beweis:

Zerlege P wie folgt:

$$P = v_0 \overset{P_{0i}}{\rightsquigarrow} v_i \overset{P_{ij}}{\rightsquigarrow} v_j \overset{P_{j\ell}}{\rightsquigarrow} v_\ell.$$

Dann gilt $w(P) = w(P_{0i}) + w(P_{ij}) + w(P_{j\ell})$.

Ist P_{ij} kein kürzester v_i-v_j-Weg, so sei Weg P'_{ij} ein kürzerer v_i-v_j-Weg.
Mit diesem erhalten wir einen v_0-v_ℓ-Weg P' von v_0 nach v_ℓ:

$$P' = v_0 \overset{P_{0i}}{\rightsquigarrow} v_i \overset{P'_{ij}}{\rightsquigarrow} v_j \overset{P_{j\ell}}{\rightsquigarrow} v_\ell.$$

Für diesen Weg gilt

$$w(P') = w(P) - w(P_{ij}) + w(P'_{ij}) < w(P). \quad \text{(Widerspruch)} \quad \square$$

5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Lemma 5.17

Sei $P = (v_0, \ldots, v_\ell)$ ein kürzester Weg von $v_0 \in V$ nach $v_\ell \in V$. Für jedes Paar i, j mit $0 \leq i \leq j \leq \ell$ ist $P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ ein kürzester Weg von v_i nach v_j.

Korollar 5.18

Sei $P = (v_0, \ldots, v_\ell)$ ein kürzester Weg von $v_0 \in V$ nach $v_\ell \in V$. Dann gilt

$$\delta(v_0, v_\ell) = \delta(v_0, v_{\ell-1}) + w(v_{\ell-1}, v_\ell).$$

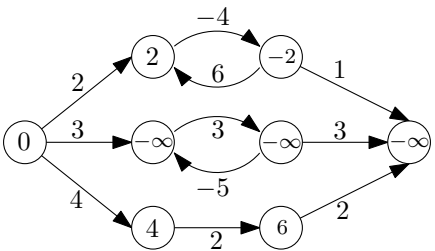
5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Im Allgemeinen sind **negative Kantengewichte** erlaubt.

5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Im Allgemeinen sind **negative Kantengewichte** erlaubt.

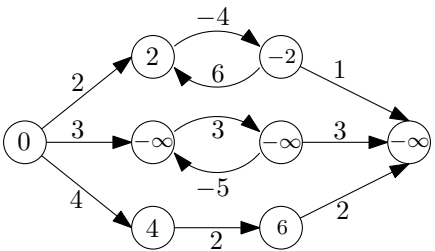
Schwierigkeit: **Kreise mit negativem Gesamtgewicht**.



5.3.1 Grundlegende Eigenschaften von kürzesten Wegen

Im Allgemeinen sind **negative Kantengewichte** erlaubt.

Schwierigkeit: **Kreise mit negativem Gesamtgewicht**.

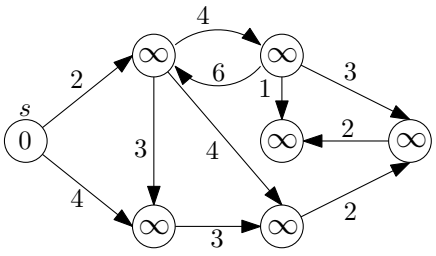


Kein negativer Kreis. $\Rightarrow$ **Kürzeste Wege einfach.**

5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

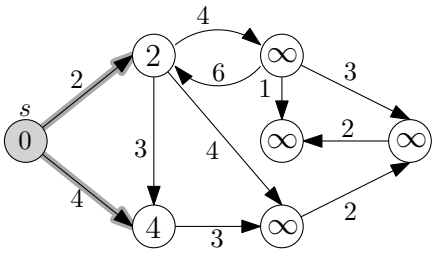
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

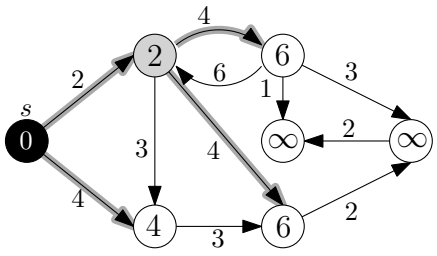
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

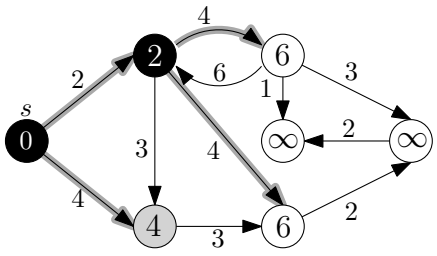
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

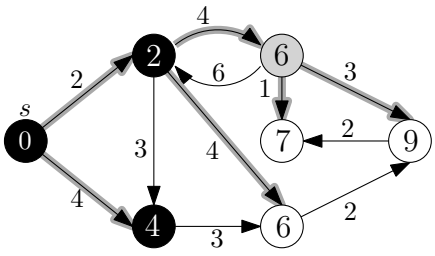
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

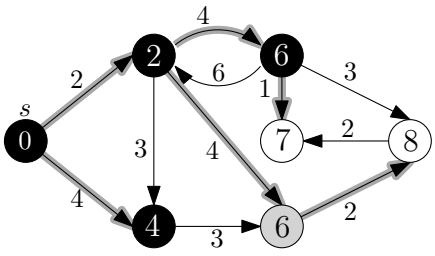
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

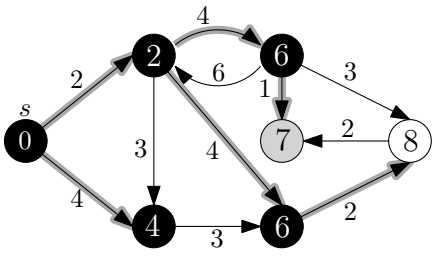
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

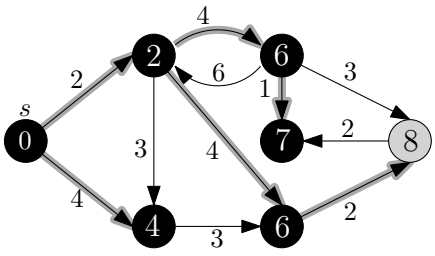
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

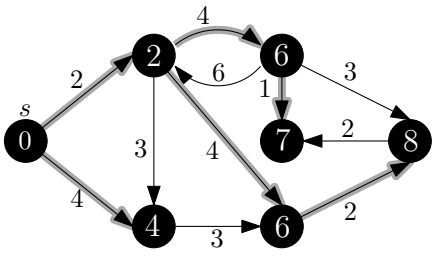
Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

```
DIJKSTRA(G, w, s)
1   for each (v ∈ V) { d(v) = ∞; π(v) = null; }
2   d(s) = 0; S = ∅;
3   while (S ≠ V) {
4       Finde u ∈ V \ S, für das d(u) minimal ist.
5       S = S ∪ {u};
6       for each ((u, v) ∈ E) {
7           if (d(v) > d(u) + w(u, v)) {
8               d(v) = d(u) + w(u, v);
9               π(v) = u;
10          }
11      }
12  }
```

Sei $G = (V, E)$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$. Jeder Knoten von s aus erreichbar. Außerdem gelte $n = |V|$ und $m = |E|$.



5.3.2 Single-Source Shortest Path Problem

Theorem 5.19

Der Algorithmus von Dijkstra terminiert auf gerichteten Graphen $G = (V, E)$ mit nichtnegativen Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$ in einem Zustand, in dem $d(v) = \delta(s, v)$ für alle $v \in V$ gilt.

Theorem 5.19

Der Algorithmus von Dijkstra terminiert auf gerichteten Graphen $G = (V, E)$ mit nichtnegativen Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$ in einem Zustand, in dem $d(v) = \delta(s, v)$ für alle $v \in V$ gilt.

Beweis:

Konvention: $w((u, v)) = \infty$ für $(u, v) \notin E$.

Theorem 5.19

Der Algorithmus von Dijkstra terminiert auf gerichteten Graphen $G = (V, E)$ mit nichtnegativen Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$ in einem Zustand, in dem $d(v) = \delta(s, v)$ für alle $v \in V$ gilt.

Beweis:

Konvention: $w((u, v)) = \infty$ für $(u, v) \notin E$.

Invariante: Am Ende jeder Iteration der while-Schleife gilt Folgendes.

1. $\forall v \in S : d(v) = \delta(s, v)$,
2. $\forall v \in V \setminus S : d(v) = \min\{\delta(s, x) + w(x, v) \mid x \in S\}$.

Theorem 5.19

Der Algorithmus von Dijkstra terminiert auf gerichteten Graphen $G = (V, E)$ mit nichtnegativen Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$ und Startknoten $s \in V$ in einem Zustand, in dem $d(v) = \delta(s, v)$ für alle $v \in V$ gilt.

Beweis:

Konvention: $w((u, v)) = \infty$ für $(u, v) \notin E$.

Invariante: Am Ende jeder Iteration der while-Schleife gilt Folgendes.

1. $\forall v \in S : d(v) = \delta(s, v)$,
2. $\forall v \in V \setminus S : d(v) = \min\{\delta(s, x) + w(x, v) \mid x \in S\}$.

Am Ende gilt $S = V$.

Die Invariante besagt dann, dass $d(v) = \delta(s, v)$ für alle Knoten $v \in S = V$ gilt.

Induktionsanfang (nach erstem Schleifendurchlauf):

Es gilt $S = \{s\}$ und $d(s) = \delta(s, s) = 0$.

Somit ist der erste Teil der Invariante erfüllt.

Induktionsanfang (nach erstem Schleifendurchlauf):

Es gilt $S = \{s\}$ und $d(s) = \delta(s, s) = 0$.

Somit ist der erste Teil der Invariante erfüllt.

Zweiter Teil der Invariante besagt:

$$\forall v \in V \setminus \{s\} : d(v) = \min\{\delta(s, x) + w(x, v) \mid x \in S\} = \delta(s, s) + w(s, v) = w(s, v).$$

Induktionsanfang (nach erstem Schleifendurchlauf):

Es gilt $S = \{s\}$ und $d(s) = \delta(s, s) = 0$.
Somit ist der erste Teil der Invariante erfüllt.

Zweiter Teil der Invariante besagt:

$$\forall v \in V \setminus \{s\} : d(v) = \min\{\delta(s, x) + w(x, v) \mid x \in S\} = \delta(s, s) + w(s, v) = w(s, v).$$

Die Gültigkeit dieser Aussage folgt daraus, dass für jeden direkten Nachfolger v von s in der for-Schleife Folgendes gesetzt wird:

$$d(v) = \min\{d(s) + w(s, v), \infty\} = w(s, v).$$

Induktionsschritt:

Wir betrachten Iteration, in der Knoten $u \in V \setminus S$ der Menge S hinzu gefügt wird.

Induktionsschritt:

Wir betrachten Iteration, in der Knoten $u \in V \setminus S$ der Menge S hinzu gefügt wird.

Wir beweisen zunächst den ersten Teil der Invariante: $\forall v \in S \cup \{u\} : d(v) = \delta(s, v)$.
Dafür genügt es für den Knoten u die Gleichung $d(u) = \delta(s, u)$ zu zeigen.

Induktionsschritt:

Wir betrachten Iteration, in der Knoten $u \in V \setminus S$ der Menge S hinzu gefügt wird.

Wir beweisen zunächst den ersten Teil der Invariante: $\forall v \in S \cup \{u\} : d(v) = \delta(s, v)$.
Dafür genügt es für den Knoten u die Gleichung $d(u) = \delta(s, u)$ zu zeigen.

Es gilt $d(u) \geq \delta(s, u)$:
Gemäß dem zweiten Teil der Invariante gilt $d(u) = \min\{\delta(s, x) + w(x, u) \mid x \in S\}$.

Induktionsschritt:

Wir betrachten Iteration, in der Knoten $u \in V \setminus S$ der Menge S hinzu gefügt wird.

Wir beweisen zunächst den ersten Teil der Invariante: $\forall v \in S \cup \{u\} : d(v) = \delta(s, v)$.
Dafür genügt es für den Knoten u die Gleichung $d(u) = \delta(s, u)$ zu zeigen.

Es gilt $d(u) \geq \delta(s, u)$:
Gemäß dem zweiten Teil der Invariante gilt $d(u) = \min\{\delta(s, x) + w(x, u) \mid x \in S\}$.
Es gilt

$$\forall x \in S : \delta(s, u) \leq \delta(s, x) + w(x, u)$$

und damit

$$\delta(s, u) \leq \min\{\delta(s, x) + w(x, u) \mid x \in S\} = d(u).$$

Es gilt $d(u) \leq \delta(s, u)$:

- Sei P ein kürzester s-u-Weg.

5.3.2 Single-Source Shortest Path Problem

Es gilt $d(u) \leq \delta(s, u)$:

- Sei P ein kürzester s-u-Weg.
- Sei y der erste Knoten auf dem Weg P, der nicht zu S gehört.
  Da $u \notin S$, muss es einen solchen Knoten y geben.

5.3.2 Single-Source Shortest Path Problem

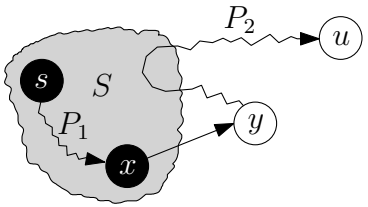
Es gilt $d(u) \leq \delta(s, u)$:

- Sei P ein kürzester s-u-Weg.
- Sei y der erste Knoten auf dem Weg P, der nicht zu S gehört.
  Da $u \notin S$, muss es einen solchen Knoten y geben.
- Weiterhin muss y einen Vorgänger haben, da der erste Knoten auf dem Weg P der Knoten $s \in S$ ist. Wir nennen diesen Vorgänger x.

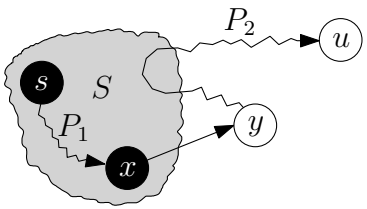
5.3.2 Single-Source Shortest Path Problem

Es gilt $d(u) \leq \delta(s, u)$:

- Sei P ein kürzester s-u-Weg.
- Sei y der erste Knoten auf dem Weg P, der nicht zu S gehört.
  Da $u \notin S$, muss es einen solchen Knoten y geben.
- Weiterhin muss y einen Vorgänger haben, da der erste Knoten auf dem Weg P der Knoten $s \in S$ ist. Wir nennen diesen Vorgänger x.
- Den Teilweg von P von s zu x nennen wir P_1 und den Teilweg von y zu u nennen wir P_2. Diese Teilwege können auch leer sein.



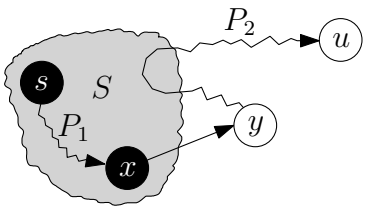
5.3.2 Single-Source Shortest Path Problem



P_1 ist kürzester s-x-Weg. Somit gilt

$$\delta(s,u) = w(P_1) + w(x,y) + w(P_2) = \delta(s,x) + w(x,y) + w(P_2) \geq d(y) + w(P_2) \geq d(y).$$

5.3.2 Single-Source Shortest Path Problem

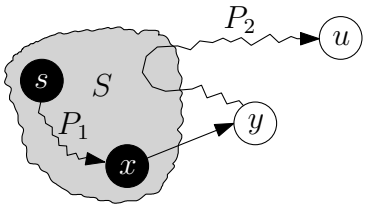


P_1 ist kürzester s-x-Weg. Somit gilt

$$\delta(s,u) = w(P_1) + w(x,y) + w(P_2) = \delta(s,x) + w(x,y) + w(P_2) \geq d(y) + w(P_2) \geq d(y).$$

Da u ein Knoten aus $V \setminus S$ mit kleinstem d-Wert ist, muss $d(u) \leq d(y)$ gelten.
Folglich gilt $\delta(s,u) \geq d(y) \geq d(u)$.

5.3.2 Single-Source Shortest Path Problem



P_1 ist kürzester s-x-Weg. Somit gilt

$$\delta(s,u) = w(P_1) + w(x,y) + w(P_2) = \delta(s,x) + w(x,y) + w(P_2) \geq d(y) + w(P_2) \geq d(y).$$

Da u ein Knoten aus $V \setminus S$ mit kleinstem d-Wert ist, muss $d(u) \leq d(y)$ gelten.
Folglich gilt $\delta(s,u) \geq d(y) \geq d(u)$.

Zusammengenommen zeigt dies den ersten Teil der Invariante: $d(u) = \delta(s,u)$.

Zweiter Teil der Invariante:

$\forall v \in V \setminus (S \cup \{u\}) : d(v) = \min\{\delta(s,x) + w(x,v) \mid x \in (S \cup \{u\})\}$.

Vorher: $\forall v \in V \setminus S : d(v) = \min\{\delta(s,x) + w(x,v) \mid x \in S\}$

Zweiter Teil der Invariante:

$\forall v \in V \setminus (S \cup \{u\}) : d(v) = \min\{\delta(s,x) + w(x,v) \mid x \in (S \cup \{u\})\}$.

Vorher: $\forall v \in V \setminus S : d(v) = \min\{\delta(s,x) + w(x,v) \mid x \in S\}$

Im Inneren der for-Schleife wird Folgendes für $v \in V$ gesetzt:

$$\begin{aligned} d(v) &= \min\{\min\{\delta(s,x) + w(x,v) \mid x \in S\}, \delta(s,u) + w(u,v)\} \\ &= \min\{\delta(s,x) + w(x,v) \mid x \in S \cup \{u\}\}. \quad \square \end{aligned}$$

Kürzeste-Wege-Bäume

Definition 5.20

Wir nennen $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$ einen **Kürzeste-Wege-Baum mit Wurzel s**, wenn die folgenden Eigenschaften erfüllt sind.

1. V' ist die Menge der Knoten, die von s aus in G erreichbar sind.
2. G' ist ein gewurzelter Baum mit Wurzel s. (Das bedeutet, dass G' azyklisch ist, selbst wenn wir die Richtung der Kanten ignorieren, und dass alle Kanten von s weg zeigen.)
3. Für alle $v \in V'$ ist der eindeutige Weg von s zu v in G' ein kürzester Weg von s nach v in G.

Kürzeste-Wege-Bäume

Definition 5.20

Wir nennen $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$ einen **Kürzeste-Wege-Baum mit Wurzel s**, wenn die folgenden Eigenschaften erfüllt sind.

1. V' ist die Menge der Knoten, die von s aus in G erreichbar sind.
2. G' ist ein gewurzelter Baum mit Wurzel s. (Das bedeutet, dass G' azyklisch ist, selbst wenn wir die Richtung der Kanten ignorieren, und dass alle Kanten von s weg zeigen.)
3. Für alle $v \in V'$ ist der eindeutige Weg von s zu v in G' ein kürzester Weg von s nach v in G.

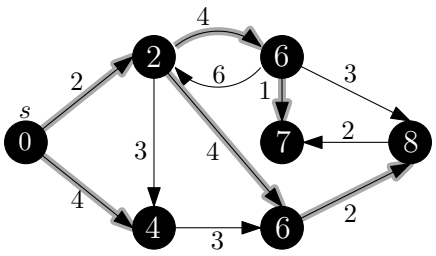
Speicherplatzbedarf: $O(n)$

Lemma 5.21

Betrachte das Ergebnis des Algorithmus von Dijkstra. Es seien

$$V_\pi = \{v \in V \mid \pi(v) \neq \textbf{null}\} \cup \{s\} \quad \text{und} \quad E_\pi = \{(\pi(v), v) \in E \mid v \in V_\pi \setminus \{s\}\}.$$

Dann ist der Graph (V_π, E_π) ein Kürzeste-Wege-Baum mit Wurzel s.



Implementierung und Laufzeit

Wir benötigen eine Datenstruktur, die die Menge $Q = V \setminus S$ verwalten kann.
Sie sollte die folgenden Operationen unterstützen:

- INSERT(x, d): Füge ein neues Element x mit Schlüssel $d \in \mathbb{R}$ in die Menge Q ein.

Implementierung und Laufzeit

Wir benötigen eine Datenstruktur, die die Menge $Q = V \setminus S$ verwalten kann.
Sie sollte die folgenden Operationen unterstützen:

- INSERT(x, d): Füge ein neues Element x mit Schlüssel $d \in \mathbb{R}$ in die Menge Q ein.
- EXTRACT-MIN$()$: Entferne aus Q ein Element mit dem kleinsten Schlüssel und gib dieses Element zurück.

Implementierung und Laufzeit

Wir benötigen eine Datenstruktur, die die Menge $Q = V \setminus S$ verwalten kann.
Sie sollte die folgenden Operationen unterstützen:

- INSERT(x, d): Füge ein neues Element x mit Schlüssel $d \in \mathbb{R}$ in die Menge Q ein.
- EXTRACT-MIN$()$: Entferne aus Q ein Element mit dem kleinsten Schlüssel und gib dieses Element zurück.
- DECREASE-KEY(x, d_1, d_2): Ändere den Schlüssel des Objektes $x \in Q$ von d_1 auf $d_2 < d_1$.

Implementierung und Laufzeit

Wir benötigen eine Datenstruktur, die die Menge $Q = V \setminus S$ verwalten kann.
Sie sollte die folgenden Operationen unterstützen:

- INSERT(x, d): Füge ein neues Element x mit Schlüssel $d \in \mathbb{R}$ in die Menge Q ein.
- EXTRACT-MIN$()$: Entferne aus Q ein Element mit dem kleinsten Schlüssel und gib dieses Element zurück.
- DECREASE-KEY(x, d_1, d_2): Ändere den Schlüssel des Objektes $x \in Q$ von d_1 auf $d_2 < d_1$.

Prioritätswarteschlangen mit EXTRACT-MIN statt EXTRACT-MAX und DECREASE-KEY statt INCREASE-KEY.

Implementierung und Laufzeit

Wir benötigen eine Datenstruktur, die die Menge $Q = V \setminus S$ verwalten kann.
Sie sollte die folgenden Operationen unterstützen:

- INSERT(x, d): Füge ein neues Element x mit Schlüssel $d \in \mathbb{R}$ in die Menge Q ein.
- EXTRACT-MIN$()$: Entferne aus Q ein Element mit dem kleinsten Schlüssel und gib dieses Element zurück.
- DECREASE-KEY(x, d_1, d_2): Ändere den Schlüssel des Objektes $x \in Q$ von d_1 auf $d_2 < d_1$.

Prioritätswarteschlangen mit EXTRACT-MIN statt EXTRACT-MAX und DECREASE-KEY statt INCREASE-KEY.

Laufzeit der Operationen: $O(\log n)$.

Theorem 5.22

Die Laufzeit des Algorithmus von Dijkstra beträgt $O((n + m) \log n)$, wenn der Graph als Adjazenzliste gegeben ist.

Theorem 5.22

Die Laufzeit des Algorithmus von Dijkstra beträgt $O((n+m)\log n)$, wenn der Graph als Adjazenzliste gegeben ist.

Beweis:

Initialisierung: $O(n \log n)$

Theorem 5.22

Die Laufzeit des Algorithmus von Dijkstra beträgt $O((n+m)\log n)$, wenn der Graph als Adjazenzliste gegeben ist.

Beweis:

Initialisierung: $O(n \log n)$

Jede Kante aus $(u, v) \in E$ wird einmal in den Zeilen 7 bis 10 betrachtet. Dann wird ggf. der Wert $d(v)$ reduziert. Dies entspricht Aufruf von DECREASE-KEY mit Laufzeit $O(\log n)$.

Theorem 5.22

Die Laufzeit des Algorithmus von Dijkstra beträgt $O((n+m)\log n)$, wenn der Graph als Adjazenzliste gegeben ist.

Beweis:

Initialisierung: $O(n \log n)$

Jede Kante aus $(u, v) \in E$ wird einmal in den Zeilen 7 bis 10 betrachtet. Dann wird ggf. der Wert $d(v)$ reduziert. Dies entspricht Aufruf von DECREASE-KEY mit Laufzeit $O(\log n)$.

Alle Aufrufe von Zeilen 7 bis 10 zusammen haben somit eine Laufzeit von $O(m \log n)$.

Theorem 5.22

Die Laufzeit des Algorithmus von Dijkstra beträgt $O((n+m)\log n)$, wenn der Graph als Adjazenzliste gegeben ist.

Beweis:

Initialisierung: $O(n \log n)$

Jede Kante aus $(u, v) \in E$ wird einmal in den Zeilen 7 bis 10 betrachtet. Dann wird ggf. der Wert $d(v)$ reduziert. Dies entspricht Aufruf von DECREASE-KEY mit Laufzeit $O(\log n)$.

Alle Aufrufe von Zeilen 7 bis 10 zusammen haben somit eine Laufzeit von $O(m \log n)$.

Außerdem n Aufrufe von EXTRACT-MIN mit Gesamtlaufzeit von $O(n \log n)$. □

5.3.3 All-Pairs Shortest Path Problem

All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.

Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

5.3.3 All-Pairs Shortest Path Problem

All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.
Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

Floyd-Warshall-Algorithmus basiert auf **dynamischer Programmierung**.
Löse für jedes Paar $i, j \in V$ und jedes $k \in \{0, \ldots, n\}$ das folgende Teilproblem:

Finde einen kürzesten i-j-Weg P_{ij}^{k}, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

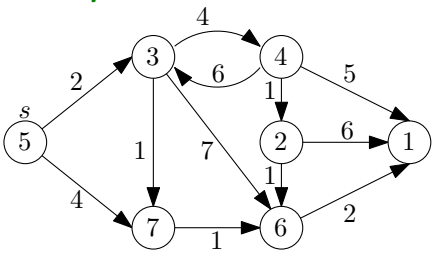
5.3.3 All-Pairs Shortest Path Problem

All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.
Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

Floyd-Warshall-Algorithmus basiert auf **dynamischer Programmierung**.
Löse für jedes Paar $i, j \in V$ und jedes $k \in \{0, \ldots, n\}$ das folgende Teilproblem:

Finde einen kürzesten i-j-Weg P_{ij}^{k}, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.



5.3.3 All-Pairs Shortest Path Problem

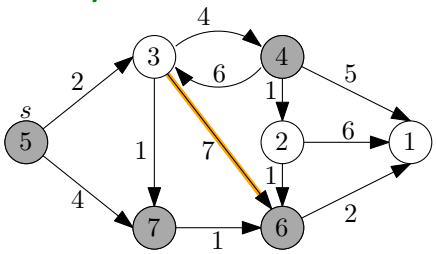
All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.
Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

Floyd-Warshall-Algorithmus basiert auf **dynamischer Programmierung**.
Löse für jedes Paar $i, j \in V$ und jedes $k \in \{0, \ldots, n\}$ das folgende Teilproblem:

Finde einen kürzesten i-j-Weg P_{ij}^{k}, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

$\delta_{3,6}^{(3)} = 7$



5.3.3 All-Pairs Shortest Path Problem

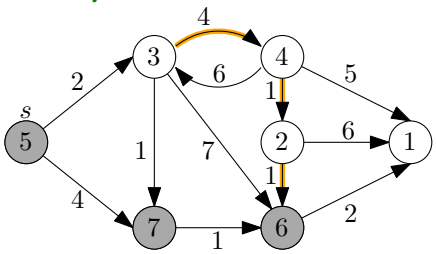
All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.
Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

Floyd-Warshall-Algorithmus basiert auf **dynamischer Programmierung**.
Löse für jedes Paar $i, j \in V$ und jedes $k \in \{0, \ldots, n\}$ das folgende Teilproblem:

Finde einen kürzesten i-j-Weg P_{ij}^{k}, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

$\delta_{3,6}^{(4)} = 6$



5.3.3 All-Pairs Shortest Path Problem

All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.

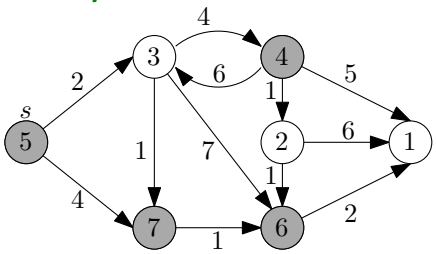
Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

Floyd-Warshall-Algorithmus basiert auf **dynamischer Programmierung**.

Löse für jedes Paar $i, j \in V$ und jedes $k \in \{0, \ldots, n\}$ das folgende Teilproblem:

Finde einen kürzesten i-j-Weg P_{ij}^{k}, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

$\delta_{5,1}^{(3)} = \infty$



5.3.3 All-Pairs Shortest Path Problem

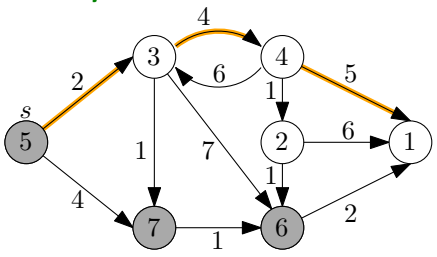
All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.
Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

Floyd-Warshall-Algorithmus basiert auf **dynamischer Programmierung**.
Löse für jedes Paar $i, j \in V$ und jedes $k \in \{0, \ldots, n\}$ das folgende Teilproblem:

Finde einen kürzesten i-j-Weg P_{ij}^{k}, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

$\delta_{5,1}^{(4)} = 11$



5.3.3 All-Pairs Shortest Path Problem

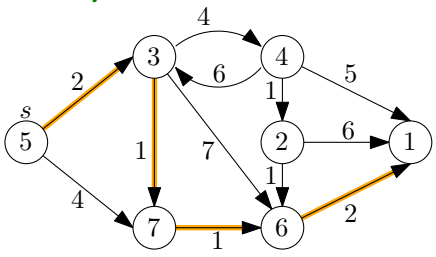
All-Pairs Shortest Path Problem (APSP)

Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gerichteter Graph mit Kantengewichten $w : E \to \mathbb{R}$.
Für einen Weg $P = (v_0, \ldots, v_\ell)$ nennen wir $v_1, \ldots, v_{\ell-1}$ die **Zwischenknoten** von P.

Floyd-Warshall-Algorithmus basiert auf **dynamischer Programmierung**.
Löse für jedes Paar $i, j \in V$ und jedes $k \in \{0, \ldots, n\}$ das folgende Teilproblem:

Finde einen kürzesten i-j-Weg P_{ij}^{k}, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

$\delta_{5,1}^{(7)} = 6$



5.3.3 All-Pairs Shortest Path Problem

Sei P_{ij}^k der kürzeste i-j-Weg, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

Sei $\delta_{ij}^{(k)}$ die Länge von P_{ij}^k.

5.3.3 All-Pairs Shortest Path Problem

Sei P_{ij}^k der kürzeste i-j-Weg, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

Sei $\delta_{ij}^{(k)}$ die Länge von P_{ij}^k.

1. Fall: P_{ij}^k enthält den **Knoten k nicht als Zwischenknoten**.

P_{ij}^k enthält nur Knoten aus $\{1, \ldots, k-1\}$ als Zwischenknoten.

5.3.3 All-Pairs Shortest Path Problem

Sei P_{ij}^k der kürzeste i-j-Weg, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

Sei $\delta_{ij}^{(k)}$ die Länge von P_{ij}^k.

1. Fall: P_{ij}^k enthält den **Knoten k nicht als Zwischenknoten**.

P_{ij}^k enthält nur Knoten aus $\{1, \ldots, k-1\}$ als Zwischenknoten.

Es gilt dann $\delta_{ij}^{(k)} = \delta_{ij}^{(k-1)}$ und $P_{ij}^k = P_{ij}^{k-1}$.

5.3.3 All-Pairs Shortest Path Problem

Sei P_{ij}^k der kürzeste i-j-Weg, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

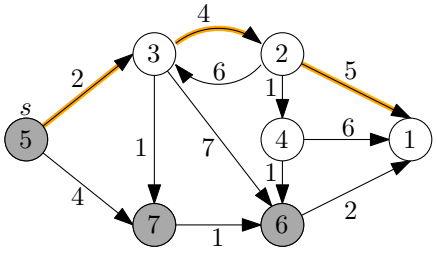
Sei $\delta_{ij}^{(k)}$ die Länge von P_{ij}^k.

1. Fall: P_{ij}^k enthält den **Knoten k nicht als Zwischenknoten**.

P_{ij}^k enthält nur Knoten aus $\{1, \ldots, k-1\}$ als Zwischenknoten.

Es gilt dann $\delta_{ij}^{(k)} = \delta_{ij}^{(k-1)}$ und $P_{ij}^k = P_{ij}^{k-1}$.

$\delta_{5,1}^{(4)} = 11$



5.3.3 All-Pairs Shortest Path Problem

Sei P_{ij}^k der kürzeste i-j-Weg, der nur Zwischenknoten aus $\{1, \ldots, k\}$ besitzt.

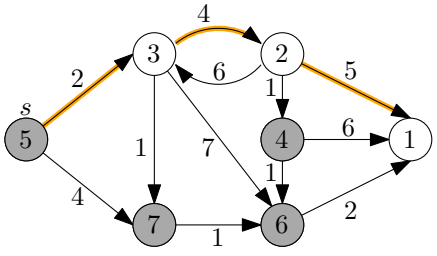
Sei $\delta_{ij}^{(k)}$ die Länge von P_{ij}^k.

1. Fall: P_{ij}^k enthält den **Knoten k nicht als Zwischenknoten**.

P_{ij}^k enthält nur Knoten aus $\{1, \ldots, k-1\}$ als Zwischenknoten.

Es gilt dann $\delta_{ij}^{(k)} = \delta_{ij}^{(k-1)}$ und $P_{ij}^k = P_{ij}^{k-1}$.

$\delta_{5,1}^{(3)} = 11$

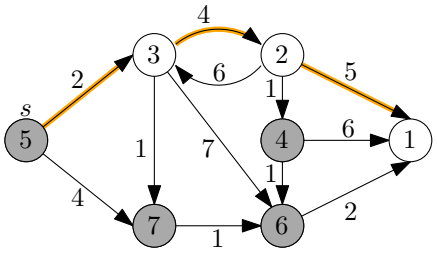


2. Fall: P_{ij}^k enthält den **Knoten k als Zwischenknoten**.

Zerlege P_{ij}^k wie folgt:

$$P_{ij}^k = i \overset{P}{\rightsquigarrow} k \overset{P'}{\rightsquigarrow} j.$$

$\delta_{5,1}^{(3)} = 11$



2. Fall: P_{ij}^k enthält den **Knoten k als Zwischenknoten**.

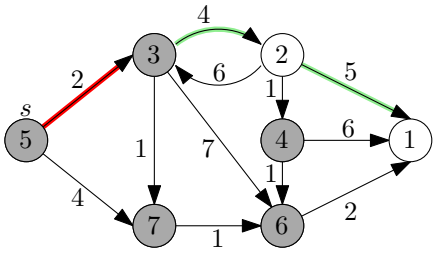
Zerlege P_{ij}^k wie folgt:

$$P_{ij}^k = i \overset{P}{\rightsquigarrow} k \overset{P'}{\rightsquigarrow} j.$$

$\delta_{5,1}^{(3)} = 11$

$\delta_{5,3}^{(2)} = 2$

$\delta_{3,1}^{(2)} = 9$

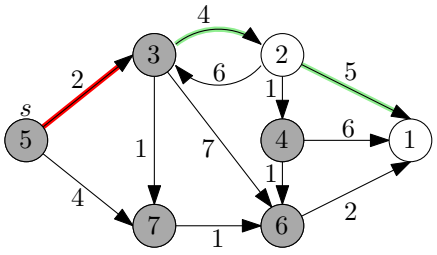


2. Fall: P_{ij}^k enthält den **Knoten k als Zwischenknoten**.

Zerlege P_{ij}^k wie folgt:

$$P_{ij}^k = i \overset{P}{\rightsquigarrow} k \overset{P'}{\rightsquigarrow} j.$$

$\delta_{5,1}^{(3)} = 11$
$\delta_{5,3}^{(2)} = 2$
$\delta_{3,1}^{(2)} = 9$



Da P_{ij}^k ein einfacher Weg ist, kommt k nur einmal vor. P und P' sind kürzeste i-k-Wege bzw. k-j-Wege, die nur Knoten aus $\{1, \ldots, k-1\}$ als Zwischenknoten benutzen:

$$\delta_{ij}^{(k)} = \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)} \quad \text{und} \quad P_{ij}^k = i \overset{P_{ik}^{k-1}}{\rightsquigarrow} k \overset{P_{kj}^{k-1}}{\rightsquigarrow} j.$$

5.3.3 All-Pairs Shortest Path Problem

Rekursionsformel:

Für alle $i, j \in V$ gilt somit

$$\delta_{ij}^{(k)} = \begin{cases} w(i,j) & \text{für } k = 0, \\ \min\{\delta_{ij}^{(k-1)}, \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)}\} & \text{für } k > 0. \end{cases}$$

Dabei sei $w(i,j) = \infty$ für $(i,j) \notin E$.

5.3.3 All-Pairs Shortest Path Problem

Rekursionsformel:

Für alle $i, j \in V$ gilt somit

$$\delta_{ij}^{(k)} = \begin{cases} w(i,j) & \text{für } k = 0, \\ \min\{\delta_{ij}^{(k-1)}, \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)}\} & \text{für } k > 0. \end{cases}$$

Dabei sei $w(i,j) = \infty$ für $(i,j) \notin E$.

Lösung des APSP:

Für alle $i, j \in V$ gilt

$$\delta(i,j) = \delta_{ij}^{(n)}.$$

5.3.3 All-Pairs Shortest Path Problem

```
FLOYD-WARSHALL(W)
1   D^(0) = W;
2   for (int k = 1; k ≤ n; k++) {
3       Erzeuge (n × n)-Nullmatrix D^(k) = (δ_ij^(k)).
4       for (int i = 1; i ≤ n; i++) {
5           for (int j = 1; j ≤ n; j++) {
6               δ_ij^(k) = min{δ_ij^(k−1), δ_ik^(k−1) + δ_kj^(k−1)};
7           }
8       }
9   }
10  return D^(n);
```

5.3.3 All-Pairs Shortest Path Problem

Theorem 5.23

Der Floyd-Warshall-Algorithmus löst das APSP für Graphen ohne Kreise mit negativem Gesamtgewicht, die als Adjazenzmatrix dargestellt sind, in Zeit $\Theta(n^3)$.

Theorem 5.23

Der Floyd-Warshall-Algorithmus löst das APSP für Graphen ohne Kreise mit negativem Gesamtgewicht, die als Adjazenzmatrix dargestellt sind, in Zeit $\Theta(n^3)$.

Negative Kreise:

G enthält negativen Kreis mit Knoten i und Knoten aus $\{1, \ldots, k\}$. $\iff \delta_{ii}^{(k)} < 0$

G enthält negativen Kreis. $\iff \exists i : \delta_{ii}^{(n)} < 0$

5.3.3 All-Pairs Shortest Path Problem

Theorem 5.23

Der Floyd-Warshall-Algorithmus löst das APSP für Graphen ohne Kreise mit negativem Gesamtgewicht, die als Adjazenzmatrix dargestellt sind, in Zeit $\Theta(n^3)$.

Negative Kreise:

G enthält negativen Kreis mit Knoten i und Knoten aus $\{1, \ldots, k\}$. $\iff \delta_{ii}^{(k)} < 0$

G enthält negativen Kreis. $\iff \exists i : \delta_{ii}^{(n)} < 0$

Vergleich zu Dijkstra:

Lösung des APSP mit Dijkstra: $O(nm \log n)$

Das ist besser für $m = o(n^2 / \log n)$.

Rekonstruktion der Pfade:

$$\delta_{ij}^{(k)} = \begin{cases} w(i,j) & \text{für } k = 0, \\ \min\{\delta_{ij}^{(k-1)}, \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)}\} & \text{für } k > 0. \end{cases}$$

Sei $\pi_{ij}^{(k)}$ der Vorgänger von j auf $P_{ij}^{(k)}$.

Rekonstruktion der Pfade:

$$\delta_{ij}^{(k)} = \begin{cases} w(i,j) & \text{für } k = 0, \\ \min\{\delta_{ij}^{(k-1)}, \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)}\} & \text{für } k > 0. \end{cases}$$

Sei $\pi_{ij}^{(k)}$ der Vorgänger von j auf $P_{ij}^{(k)}$.

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

Rekonstruktion der Pfade:

$$\delta_{ij}^{(k)} = \begin{cases} w(i,j) & \text{für } k = 0, \\ \min\{\delta_{ij}^{(k-1)}, \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)}\} & \text{für } k > 0. \end{cases}$$

Sei $\pi_{ij}^{(k)}$ der Vorgänger von j auf $P_{ij}^{(k)}$.

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

Ist $\delta_{ij}^{(k)} < \delta_{ij}^{(k-1)}$, so gilt $P_{ij}^{k} = i \overset{P_{ik}^{k-1}}{\rightsquigarrow} k \overset{P_{kj}^{k-1}}{\rightsquigarrow} j$.

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{falls } \delta_{ij}^{(k)} = \delta_{ij}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{falls } \delta_{ij}^{(k)} < \delta_{ij}^{(k-1)}. \end{cases}$$