2.6 Graphen

2.6.4 Kürzeste Pfade

Kürzeste Pfade

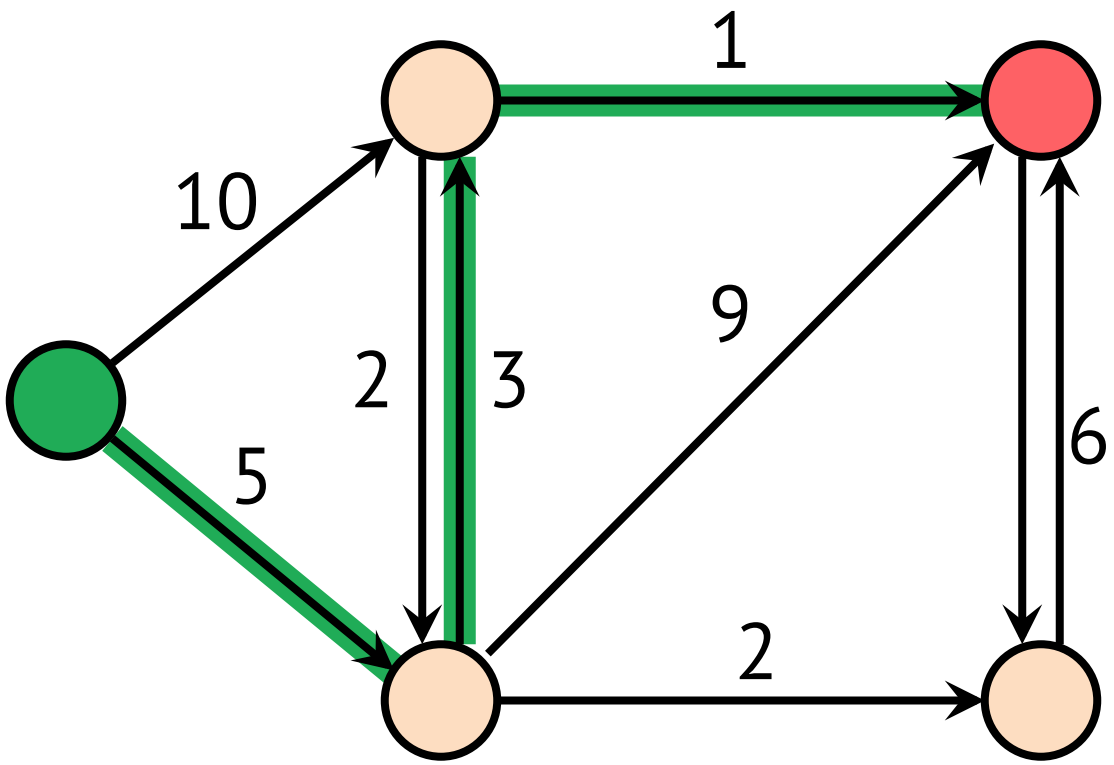
- Gegeben sei ein gerichteter, gewichteter Graph $G=(V,E)$ mit Gewichtsfunktion $w:E \rightarrow R$. Das Gewicht $w(p)$ eines Pfades

$$p = v_0, v_1, \ldots, v_k$$

ist definiert als

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Kürzeste Pfade



$$w(p) = 5 + 3 + 1 = 9$$

Kürzeste Pfade

- Das kürzeste Pfad-Gewicht $\delta(u,v)$ von Knoten u zu Knoten v ist definiert als

  $\delta(u,v) = \min\{w(p): p \text{ ist Pfad von u nach v}\}$

  wobei $\min \varnothing = \infty$

- Ein kürzester Pfad von Knoten u zu Knoten v ist ein beliebiger Pfad p mit

$$w(p) = \delta(u,v)$$

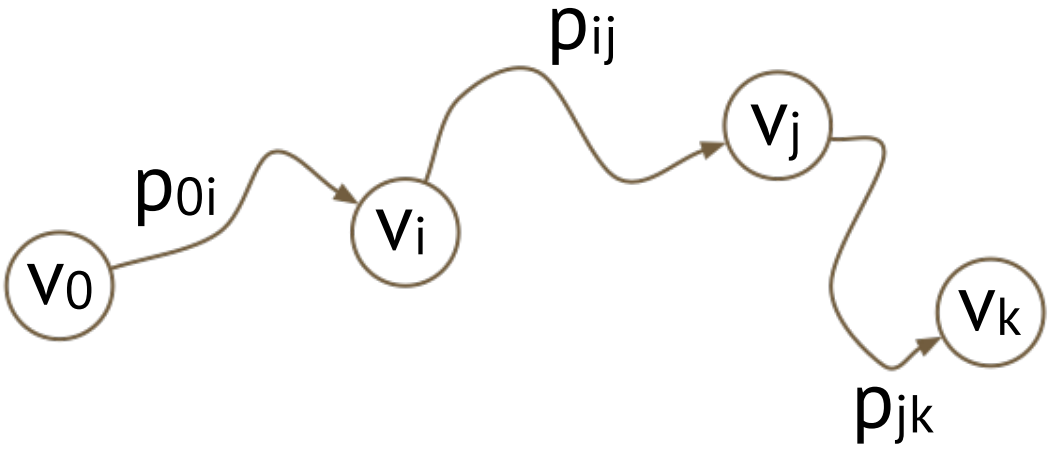
Kürzeste Pfade

- Lemma

  Teilpfade von kürzesten Pfaden sind ebenfalls kürzeste Pfade.

  Ist $p=v_0,\ldots,v_k$ ein kürzester Pfad von v_0 nach v_k, so ist für $0 \leqslant i < j \leqslant k$ der Teilpfad $p_{ij}=v_i,\ldots,v_j$ ein kürzester Pfad von v_i nach v_j

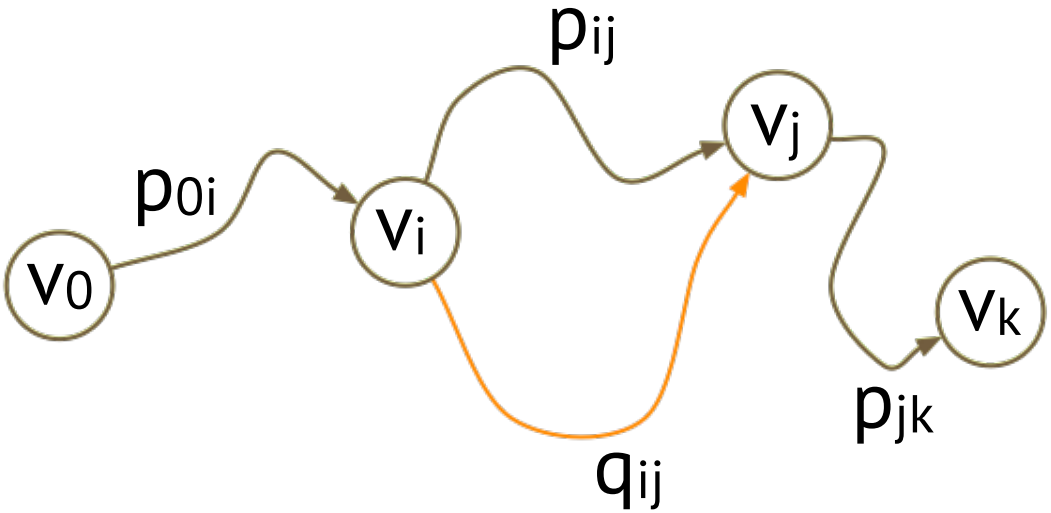
- Beweis



Es ist $p = p_{0i}, p_{ij}, p_{jk}$ und damit
$w(p) = w(p_{i0}) + w(p_{ij}) + w(p_{jk})$

Kürzeste Pfade

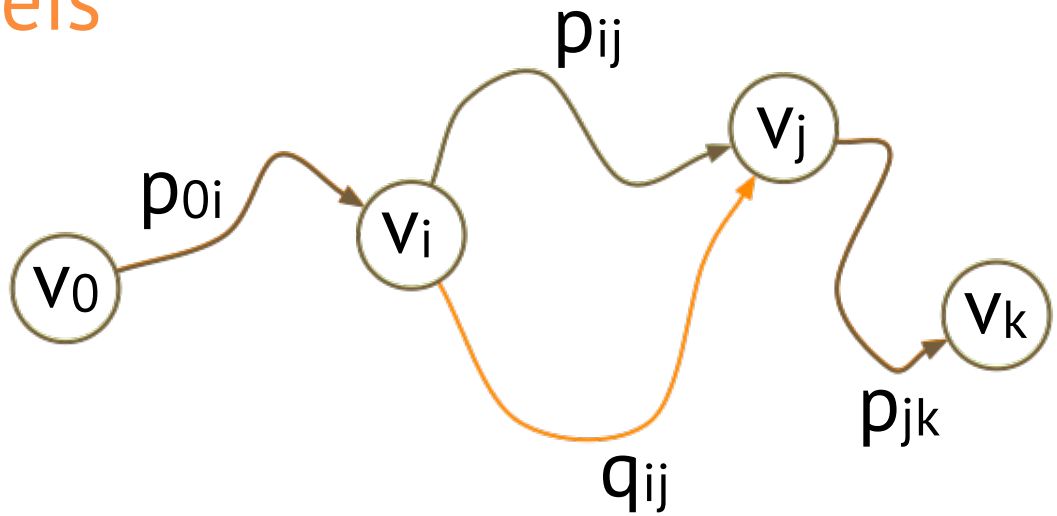
- Beweis



Annahme: Es gibt einen Pfad q_{ij} von v_i nach v_j mit $w(q_{ij})<w(p_{ij})$

Kürzeste Pfade

- Beweis



$$\begin{aligned} w(p') &= w(p_{0i}) + w(p_{ij}) + w(p_{jk}) \\ &< w(p_{0i}) + w(q_{ij}) + w(p_{jk}) \\ &= w(p) \end{aligned}$$

- Für den Pfad $p'=p_{0i},q_{ij},p_{jk}$ gilt

Widerspruch!

Kürzeste Pfade

- Single-source shortest-paths Problem

  Gegeben sei ein Graph $G=(V,E)$ und ein Startknoten $s \in V$. Bestimme zu jedem Knoten $u \in V$ einen kürzesten Pfad von s nach u.

- All-pairs shortest-paths Problem

  Gegeben sei ein Graph $G=(V,E)$. Bestimme zu jedem Knotenpaar $u \in V$ und $v \in V$ einen kürzesten Pfad von u nach v.

Darstellung

- Vorgänger

  Ein Pfad von s nach v wird dargestellt, indem jeder Knoten des Pfades seinen Vorgänger in einem Attribut p speichert.

Darstellung

- Oberschranke
  In jedem Knoten v des Graphen wird ein Attribut d gespeichert, das eine obere Schranke für das Gewicht des kürzesten Pfades von s nach v darstellt, d.h. es gilt immer die Invariante

$$\delta(s,v) \leqslant d[v]$$

- Ein Knoten heißt final, falls die Oberschranke minimal ist, d.h. falls

$$\delta(s,v) = d[v]$$

Darstellung

- InitializeSingleSource(G,s)

  for each $v \in V$ do

    $d[v] \leftarrow \infty$

    $p[v] \leftarrow$ NIL

  $d[s] \leftarrow 0$

Darstellung

- Relaxation einer Kante (u,v)

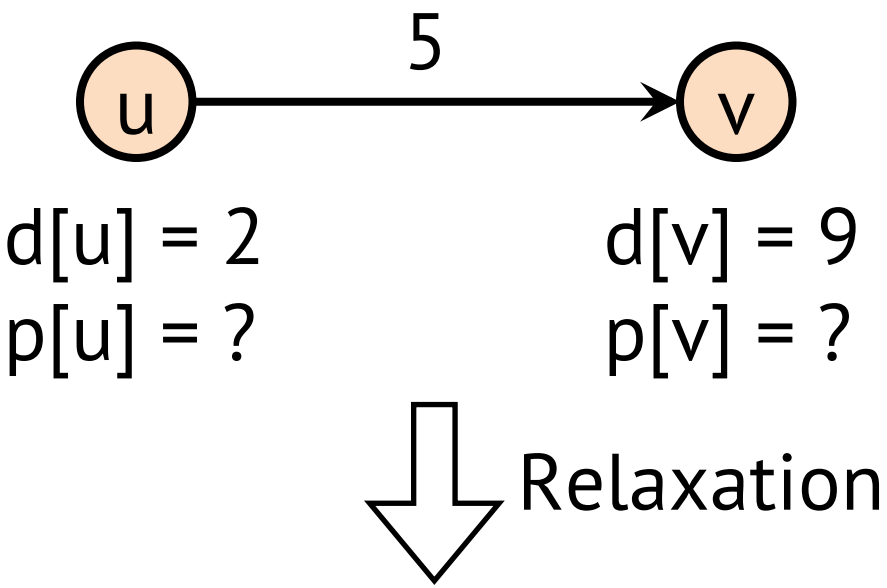
  Die Oberschranke d[v] kann verbessert werden, wenn eine Kante von u nach v existiert, so dass

  $$d[u] + w(u,v) < d[v]$$

  ist. In diesem Fall setzt man

  $$d[v] \leftarrow d[u] + w(u,v)$$
  $$p[v] \leftarrow u$$

Relaxation



Relaxation

- Relax(u,v)

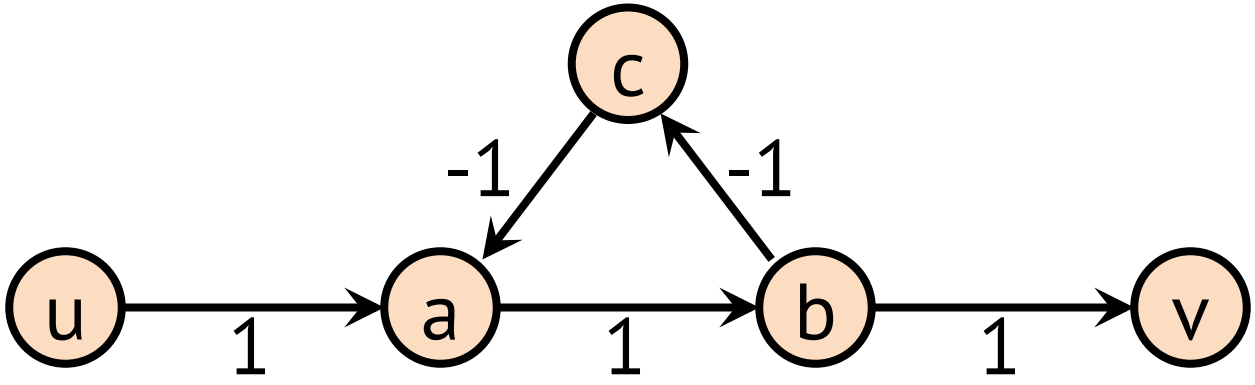
  if d[v] > d[u] + w(u,v) then

  d[v] ← d[u] + w(u,v)

  p[v] ← u

Negative Gewichte

- Prinzipiell sind gewichtete Graphen mit negativen Gewichten zulässig, dadurch kann aber die Frage nach kürzesten Pfaden unsinnig werden.



2.6.4 Kürzeste Pfade

2.6.4.1 Definition und Eigenschaften

2.6.4.2 Dijkstras Algorithmus

2.6.4.3 Floyd-Warshall Algorithmus

2.6.4.4 Transitive Hülle

Dijkstras Algorithmus

- Dijkstras Algorithmus löst das single-source shortest-path Problem für den Fall, dass alle Kantengewichte positiv sind.

Dijkstras Algorithmus

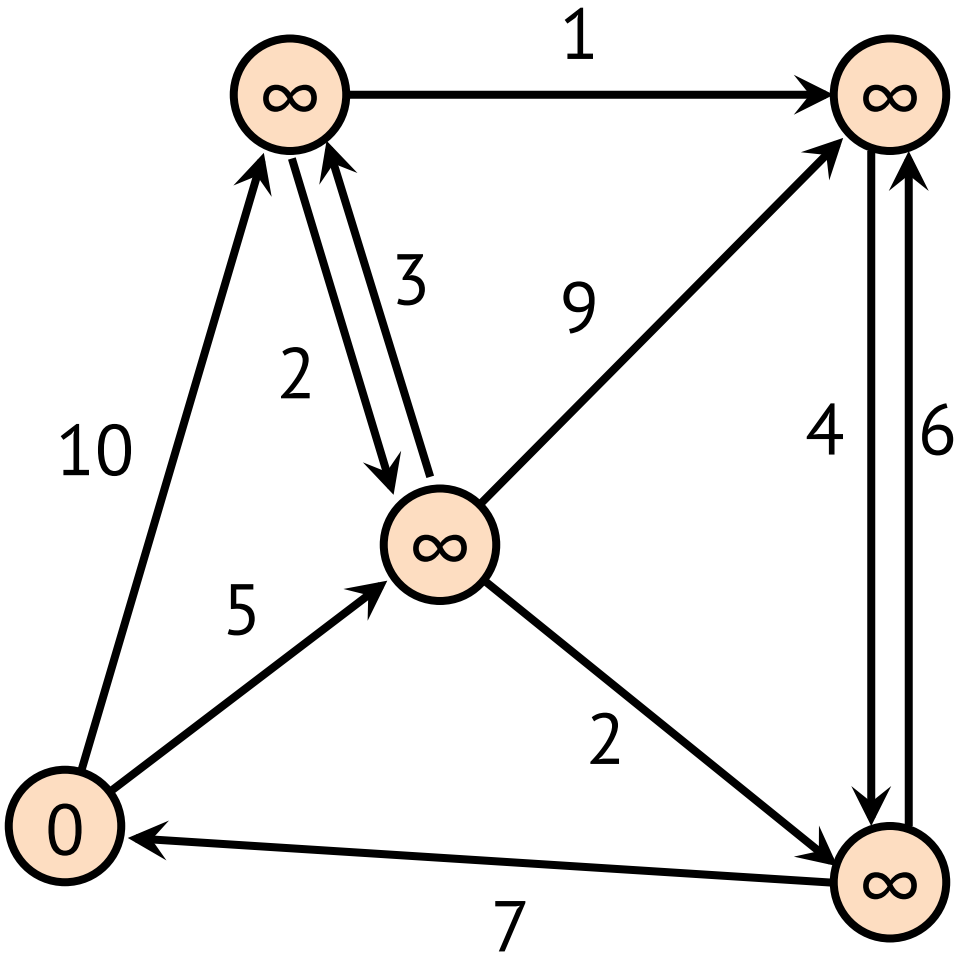
1. Es sei S eine anfangs leere Menge von Knoten.
2. Wähle einen Knoten $v \in V - S$ mit minimaler Oberschranke $d[v]$.
3. Füge v in S ein und relaxiere alle von v ausgehenden Kanten.
4. Gehe zu Schritt 2 falls $V - S \neq \emptyset$

Algorithmus

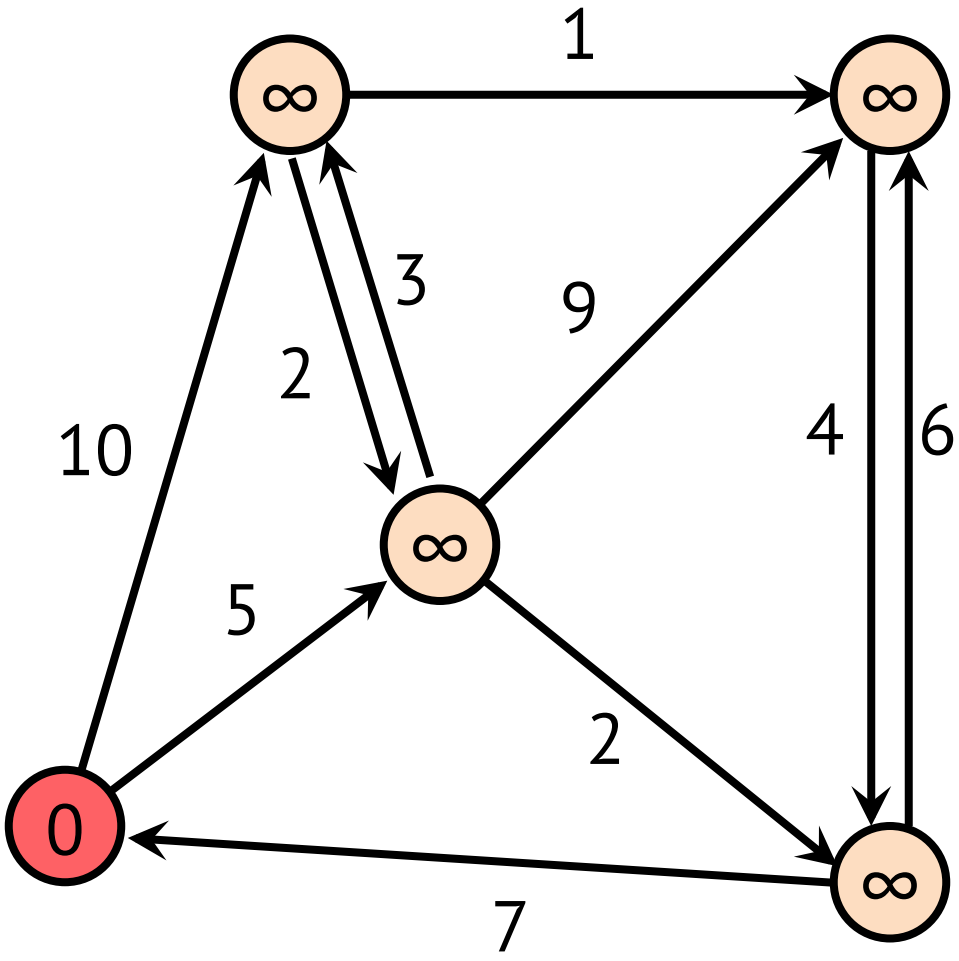
- Dijkstra(G,s)

```
    InitializeSingleSource(G,s)
    S ← ∅
    Q ← V
    while Q ≠ ∅ do
        u ← ExtractMinimum(Q)
        S ← S ∪ {u}
        for each v ∈ Adj[u] do
            Relax(u,v)
```

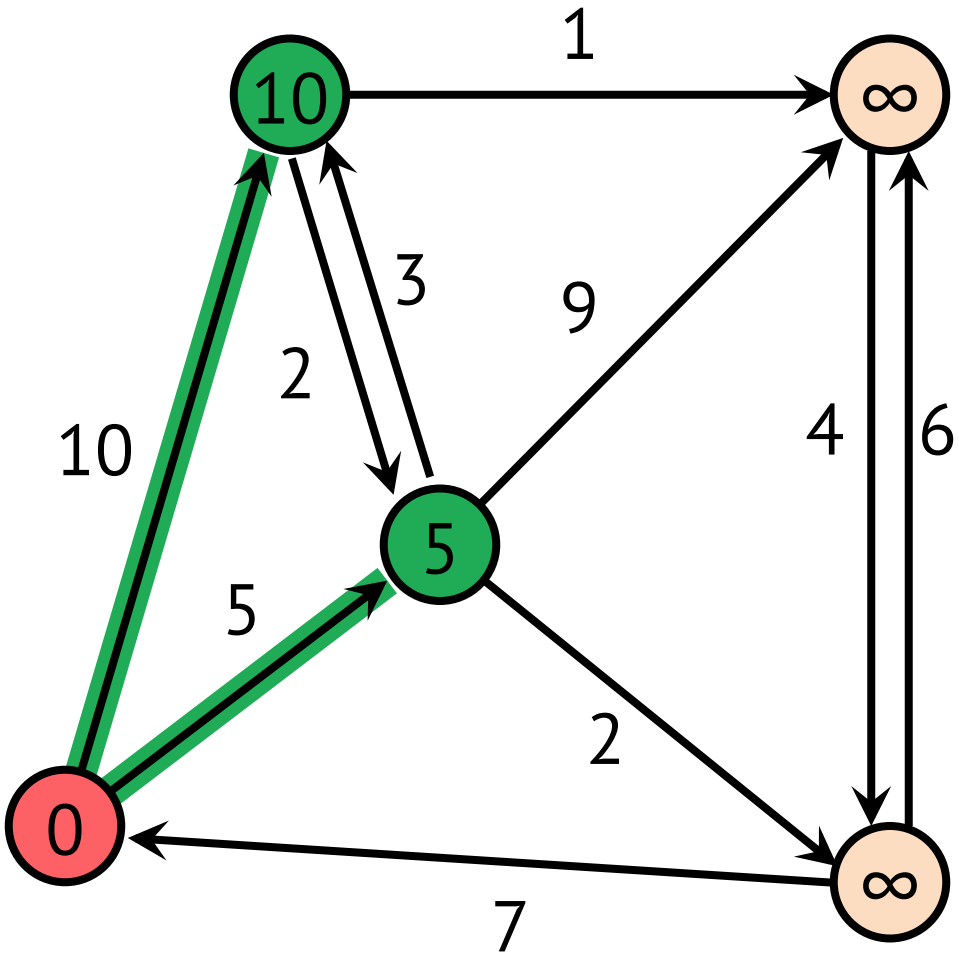
Beispiel



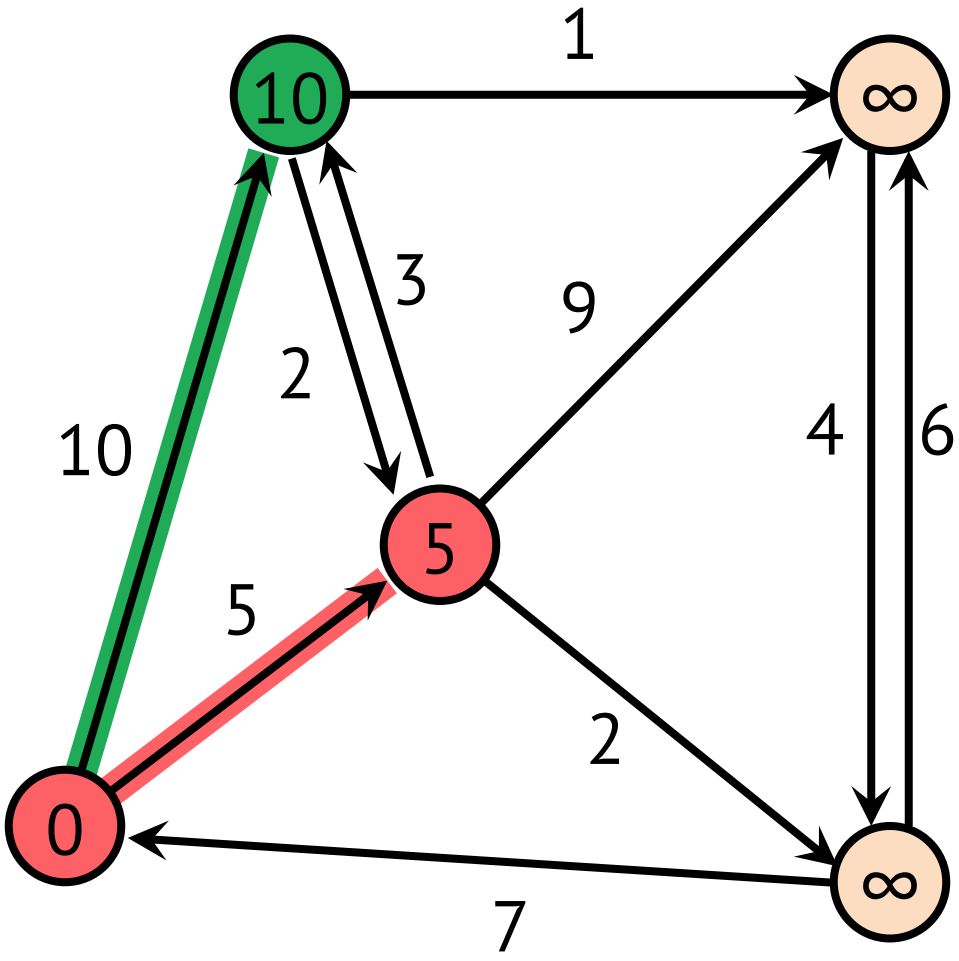
Beispiel



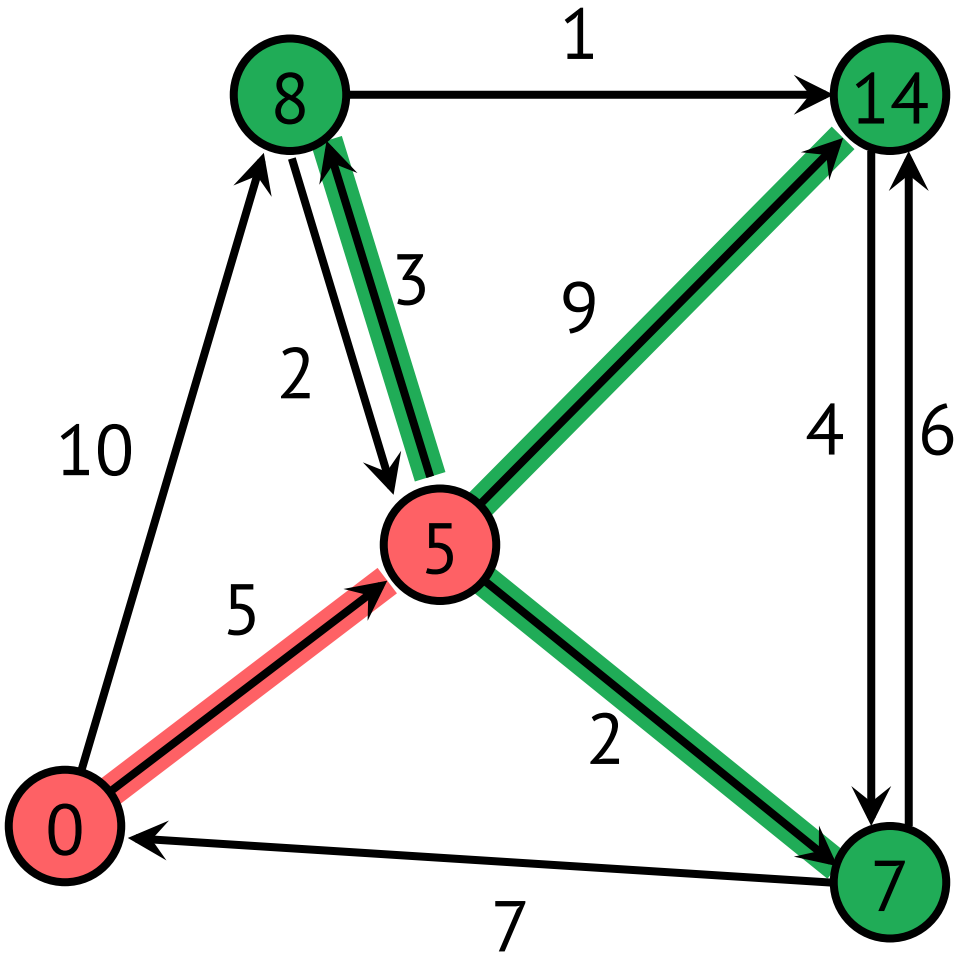
Beispiel



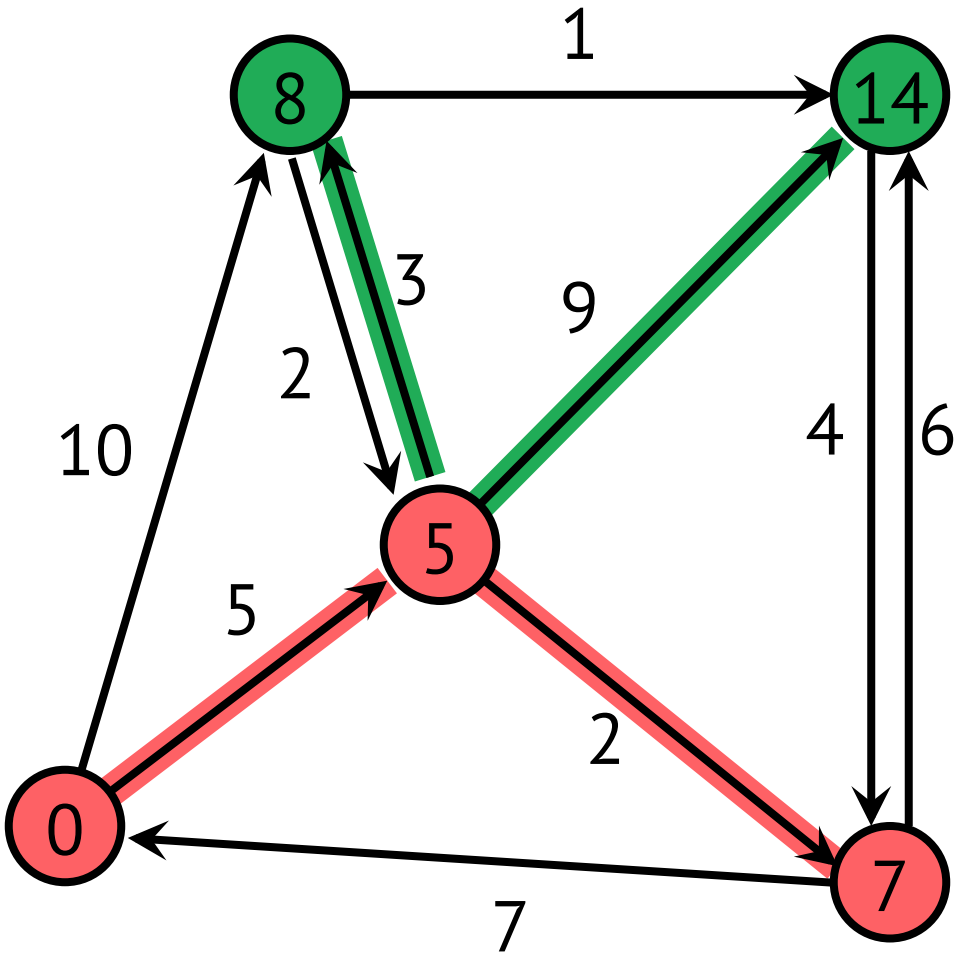
Beispiel



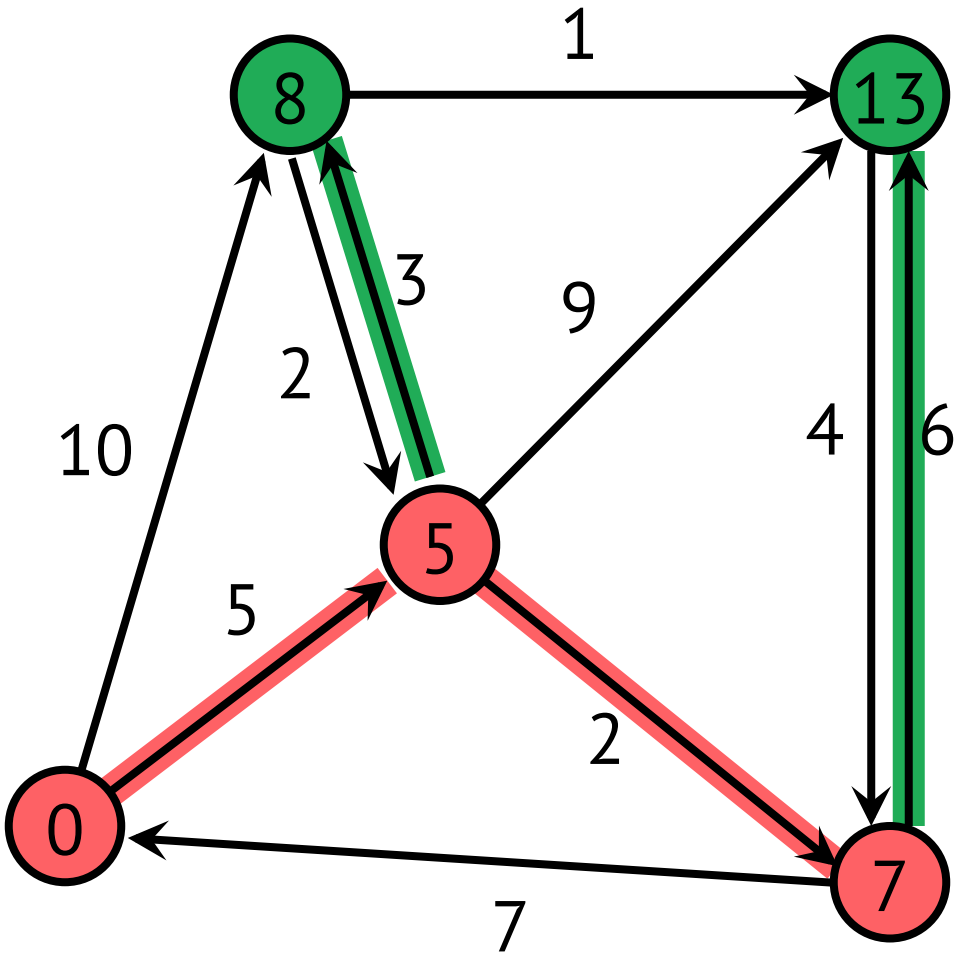
Beispiel



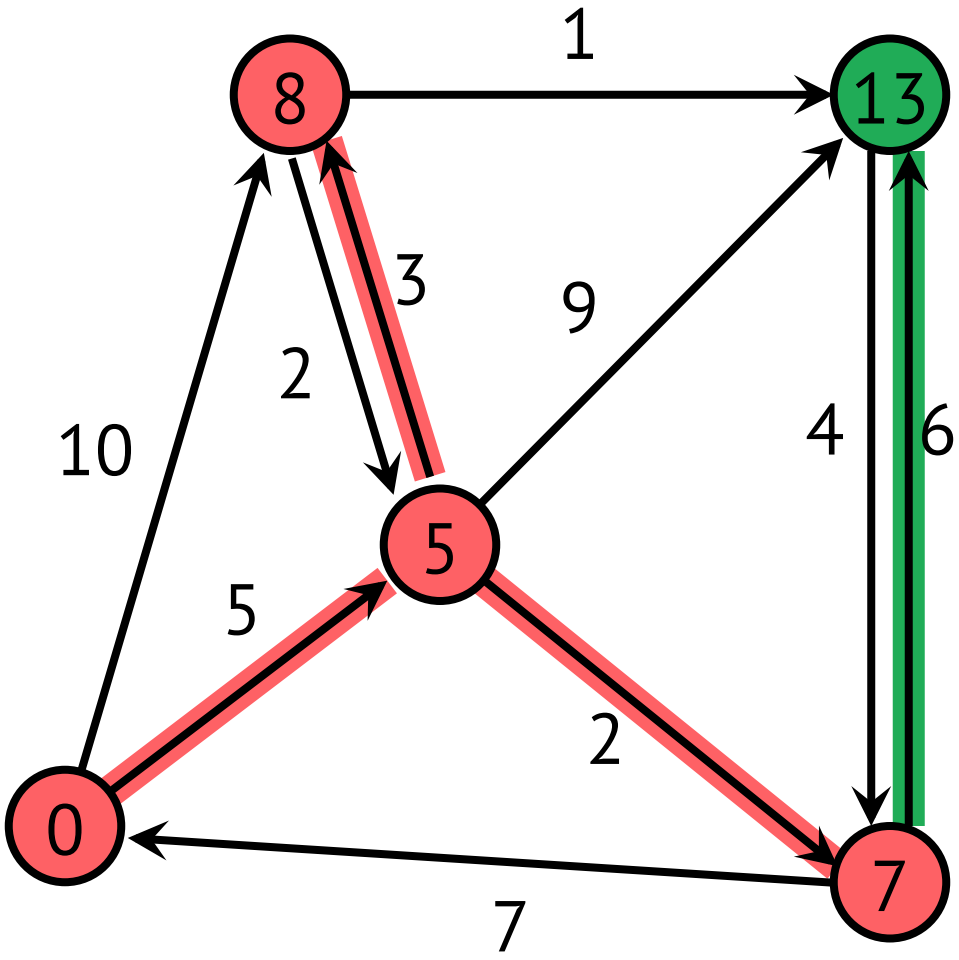
Beispiel



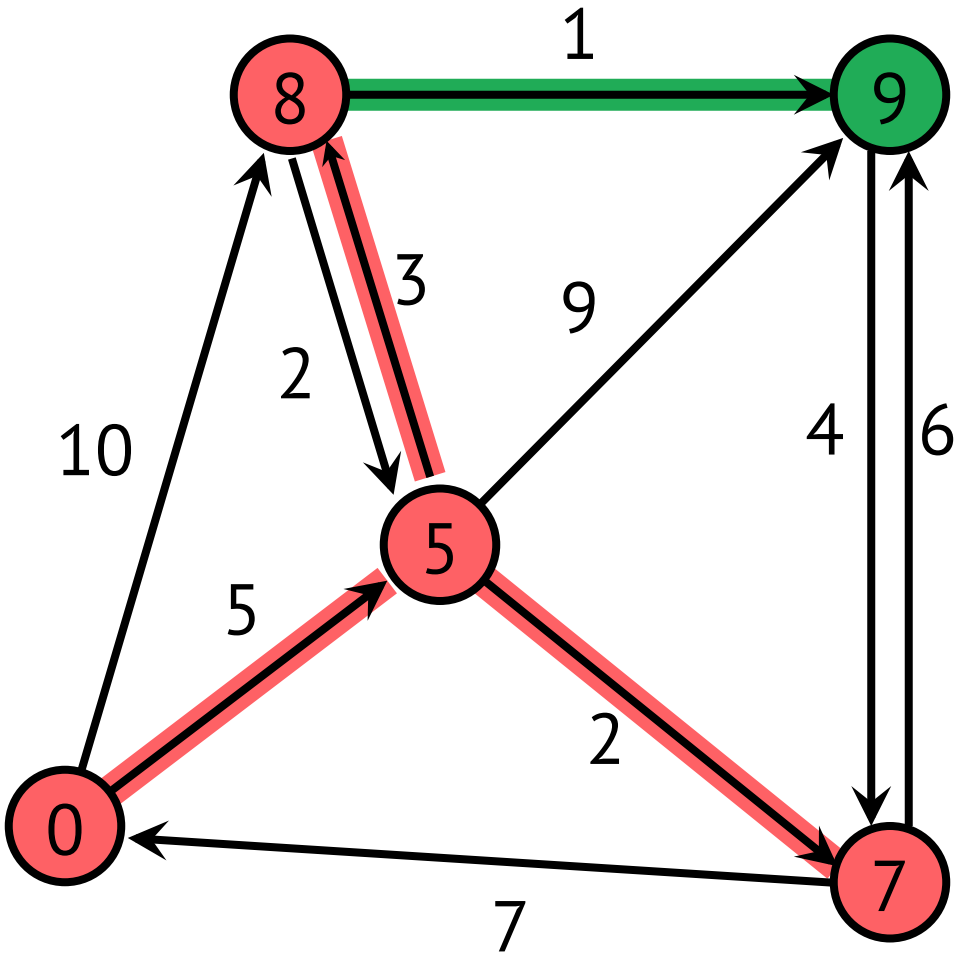
Beispiel



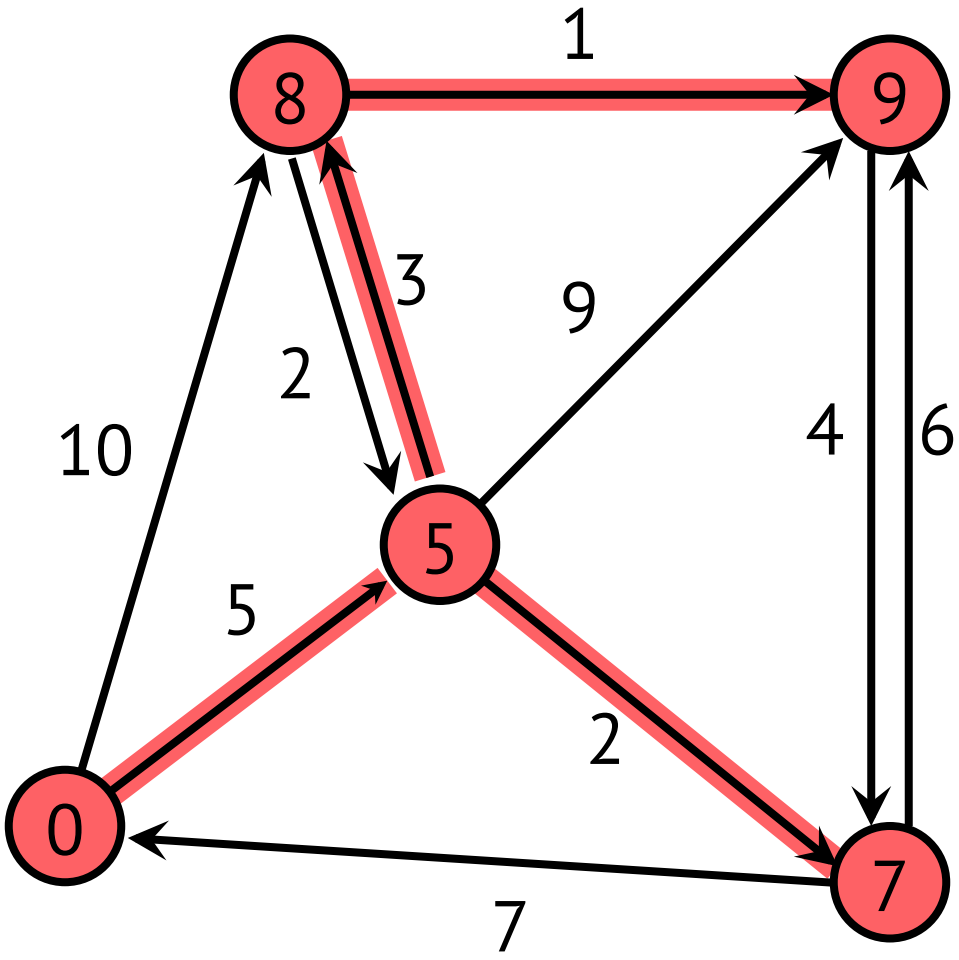
Beispiel



Beispiel



Beispiel



Korrektheit

- Korrektheit von Dijkstras Algorithmus

  Es sei $G=(V,E)$ ein gerichteter, gewichteter Graph mit nichtnegativer Gewichtsfunktion w und $s \in V$ ein Startknoten. Dann gilt: Dijkstras Algorithmus angewandt auf G terminiert mit $d[v] = \delta(s,v)$ für alle Knoten $v \in V$.

Korrektheit

- Beweis
  - Schleifeninvariante

    Es gilt $d[v] = \delta(s,v)$ für alle Knoten $v \in S$,

    d.h. alle Knoten in S sind final.
  - Initialisierung

    Vor Ausführung der Schleife ist $S = \varnothing$ und

    die Schleifeninvariante gilt trivialerweise.

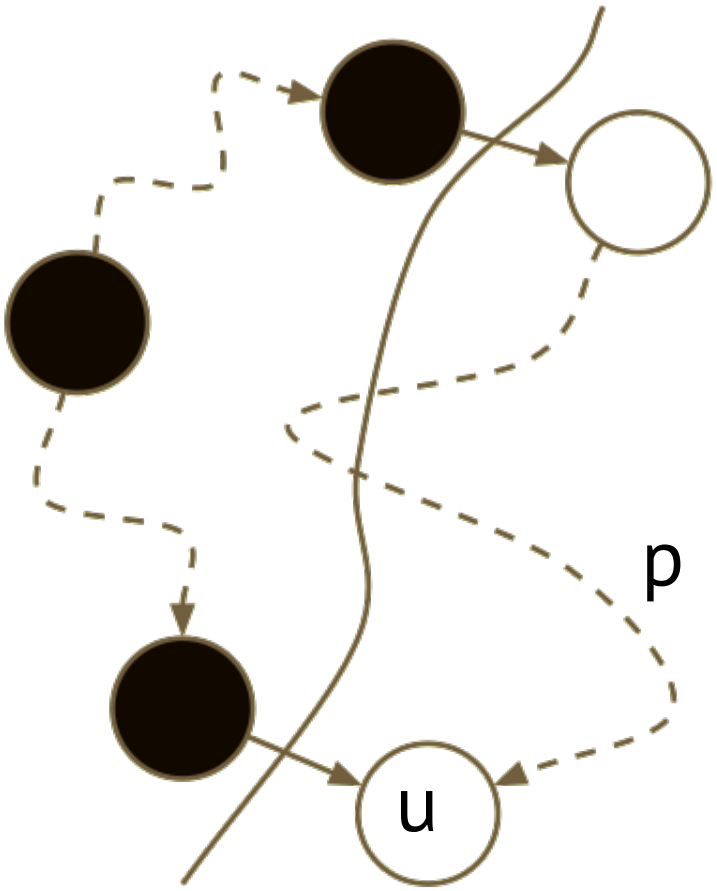
Korrektheit

- Beweis
  - Schleife

    Behauptung: Es gilt d[v]= δ (s,v) für jeden Knoten v der zu S hinzugefügt wird.
  - Beweis durch Widerspruch: Es sei u der erste Knoten, für den d[u]≠ δ (s,u) ist...

Korrektheit

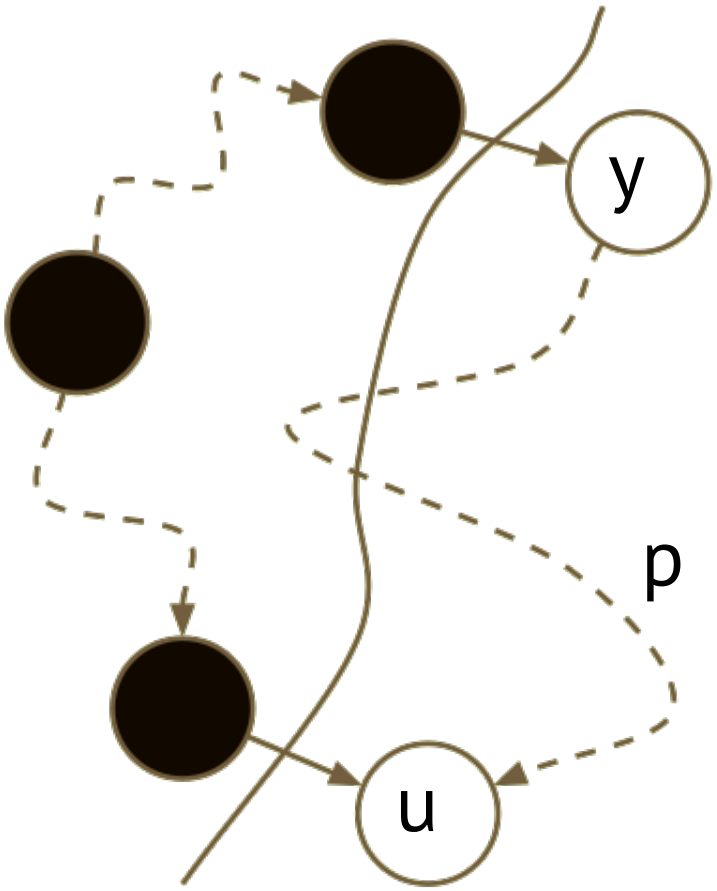
Es sei u der erste Knoten, für den d[u]≠δ(s,u) ist...



u ist von s erreichbar, es gibt also einen kürzesten Pfad p von s nach u

Korrektheit

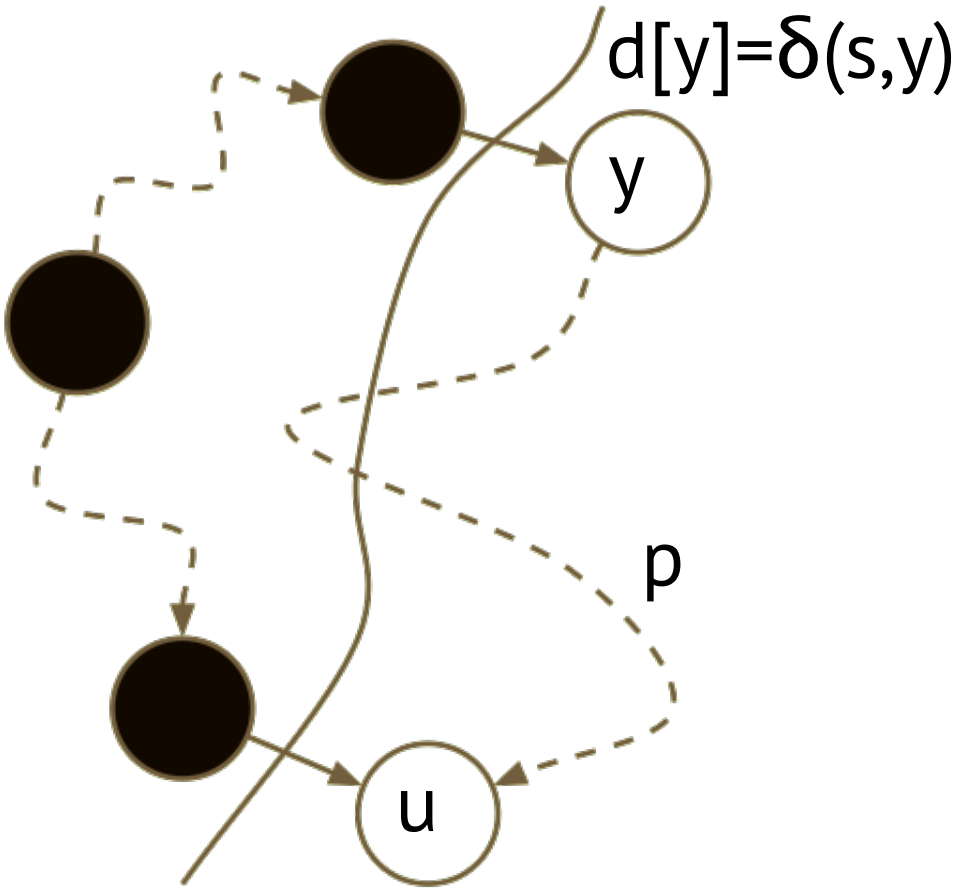
Es sei u der erste Knoten, für den $d[u] \neq \delta(s,u)$ ist...



Es sei y der erste Knoten auf p, der nicht in S liegt und x sei sein Vorgänger.

Korrektheit

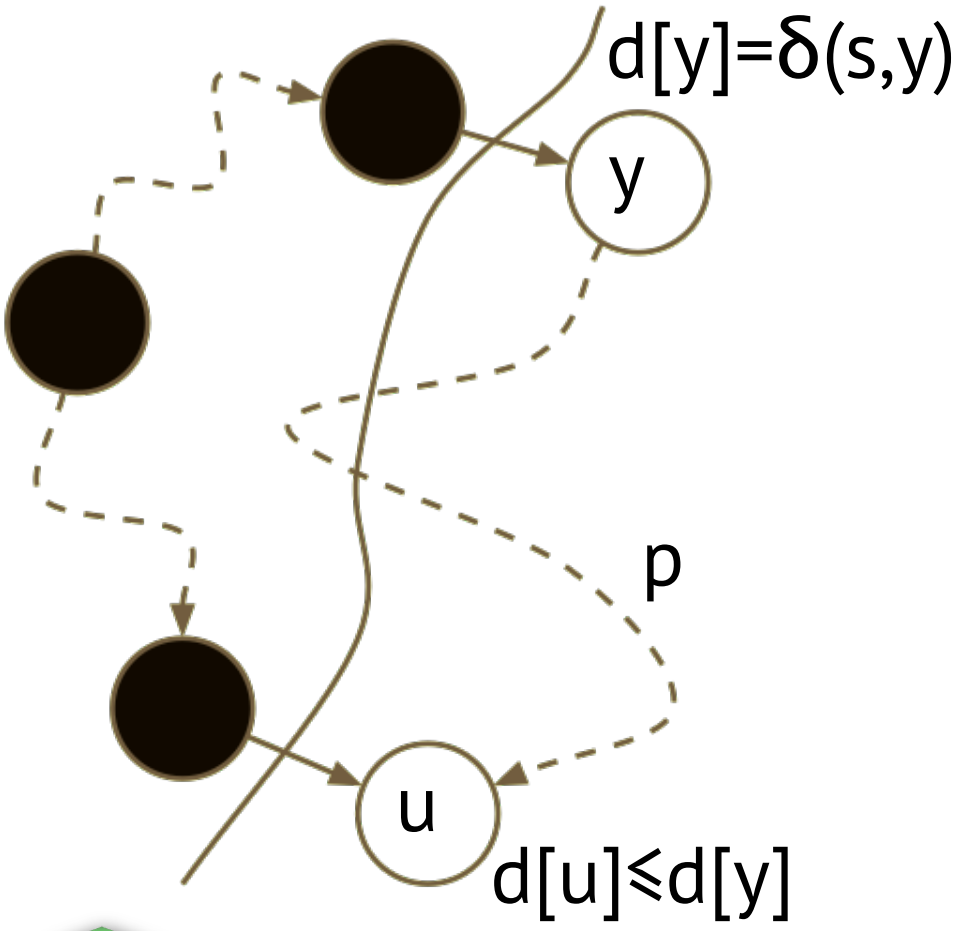
Es sei u der erste Knoten, für den d[u]≠δ(s,u) ist...



Teilpfade von kürzesten Pfaden sind kürzeste Pfade, (x,y) wurde bereits relaxiert, also ist d[y]=δ(s,y)

Korrektheit

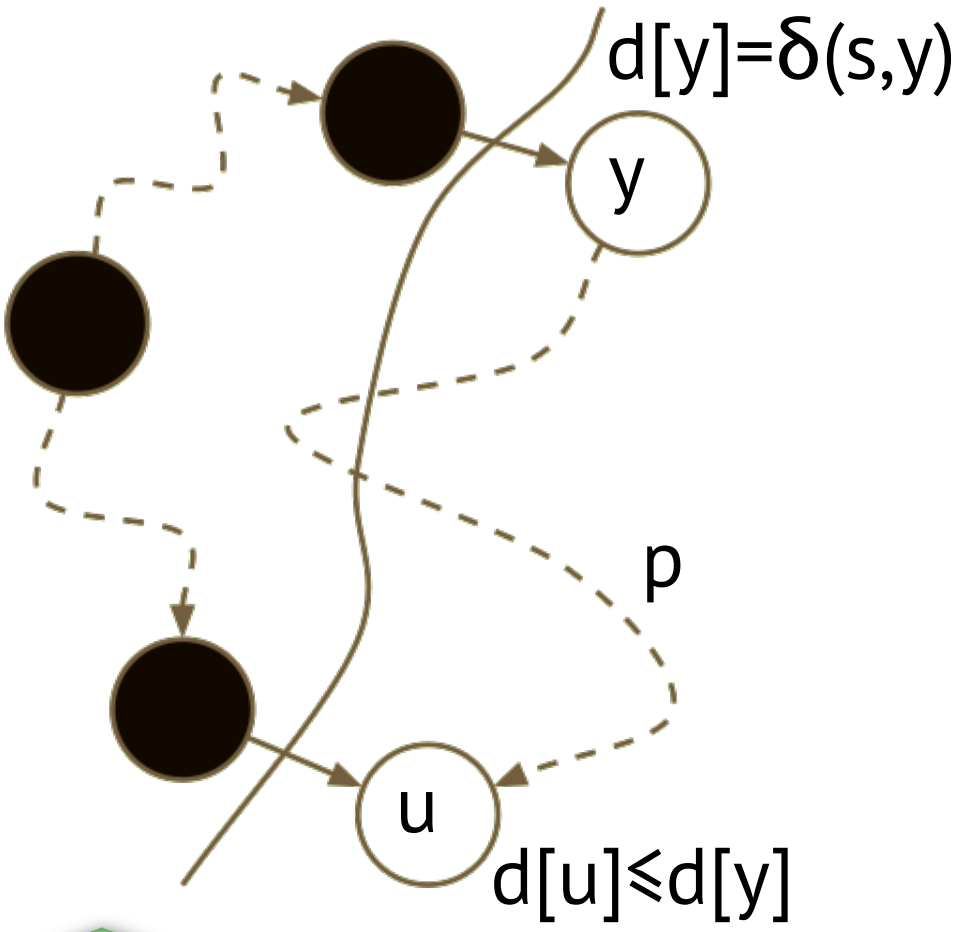
Es sei u der erste Knoten, für den $d[u] \neq \delta(s,u)$ ist...



u wird von Dijkstras Algorithmus vor y ausgewählt, also ist $d[u] \leqslant d[y]$

Korrektheit

Es sei u der erste Knoten, für den $d[u] \neq \delta(s,u)$ ist...



Es gilt

$d[y]=\delta(s,y) \leqslant \delta(s,u) \leqslant d[u]$

und

$d[u] \leqslant d[y]$

also

$d[y]=\delta(s,y)=\delta(s,u)=d[u]$

und insbesondere

$d[u]=\delta(s,u)$

Widerspruch!

Korrektheit

- Beweis
  - Terminierung

    Terminierung wenn $Q=\varnothing$, also wegen

    $Q = V-S$, wenn $S=V$

Analyse

- Aufwand

  Implementierung der Priority-Queue durch einen binären Heap.

Analyse

- Dijkstra(G,s)

```
    InitializeSingleSource(G,s)
    S ← ∅                                   Initialisierung: O(V)
    Q ← V
    while Q ≠ ∅ do
        u ← ExtractMinimum(Q)
        S ← S ∪ {u}
        for each v ∈ Adj[u] do
            Relax(u,v)
```

Analyse

- Dijkstra(G,s)

```
InitializeSingleSource(G,s)
S ← ∅
Q ← V
while Q ≠ ∅ do
    u ← ExtractMinimum(Q)
    S ← S ∪ {u}
    for each v ∈ Adj[u] do
        Relax(u,v)
```

O(V)

O(logV)

Aufwand für binären Heap

Analyse

- Dijkstra(G,s)

```
InitializeSingleSource(G,s)
S ← ∅
Q ← V
while Q ≠ ∅ do
    u ← ExtractMinimum(Q)
    S ← S ∪ {u}
    for each v ∈ Adj[u] do
        Relax(u,v)
```

Schleife wird V mal durchlaufen, insgesamt also O (V×logV)

Analyse

- Dijkstra(G,s)
  - InitializeSingleSource(G,s)
  - S ← ∅
  - Q ← V
  - while Q ≠ ∅ do
    - u ← ExtractMinimum(Q)
    - S ← S ∪ {u}
    - for each v ∈ Adj[u] do
      - Relax(u,v)

Jede Kante wird höchstens einmal relaxiert, insgesamt also O(E×logV)

Analyse

- Dijkstra(G,s)

```
    InitializeSingleSource(G,s)        ┐
    S ← ∅                              │  O(V)
    Q ← V                              ┘
    while Q ≠ ∅ do                     ┐
        u ← ExtractMinimum(Q)          │
        S ← S ∪ {u}                    │  O((E+V)×log V)
        for each v ∈ Adj[u] do         │
            Relax(u,v)                 ┘
```

Analyse

- Fazit

  Falls G zusammenhängend ist, so hat Dijkstras Algorithmus einen Aufwand von

$$O(E \times \log V)$$

2.6.4 Kürzeste Pfade

Kürzeste Pfade

- Single-source shortest-paths Problem

  Gegeben sei ein Graph G=(V,E) und ein Startknoten $s \in V$. Bestimme zu jedem Knoten $u \in V$ einen kürzesten Pfad von s nach u.

- All-pairs shortest-paths Problem

  Gegeben sei ein Graph G=(V,E). Bestimme zu jedem Knotenpaar $u \in V$ und $v \in V$ einen kürzesten Pfad von u nach v.

Kürzeste Pfade

- In diesem Abschnitt gehen wir davon aus, dass …
  - … die Knotenmenge V des Graphen oBdA $V=\{1,2,\ldots,n\}$ ist.
  - … die gewichteten Kanten in einer Adjazenzmatrix $W=(w_{ij})$ gespeichert sind.

Floyd-Warshall Algorithmus

- Es sei $v_0, v_2, \ldots, v_{m-1}, v_m$ ein einfacher Pfad. Die Knoten $v_1, \ldots, v_{m-1}$ heißen Zwischenknoten des Pfades.
- Es sei P_k die Menge der Pfade, deren sämtliche Zwischenknoten in $\{1,\ldots,k\}$ liegen.

$$p_{ij} \in P_k$$

enthält nur Zwischenknoten v mit $1 \leqslant v \leqslant k$

Floyd-Warshall Algorithmus

- Idee: Dynamisches Programmieren
  - Rekursionsformel für die Gewichte der kürzesten Pfade d
  - Rekursionsformel für die Vorgängerattribute p

Floyd-Warshall Algorithmus

- Rekursionsformel für die Gewichte

  Es sei $d_k[i,j]$ das Gewicht eines kürzesten Pfades von Knoten i nach Knoten j in P_k. Insbesondere ist also $d_n[i,j]$ das Gewicht eines kürzesten Pfades von i nach j in G.

Floyd-Warshall Algorithmus

- Initialisierung $k = 0$

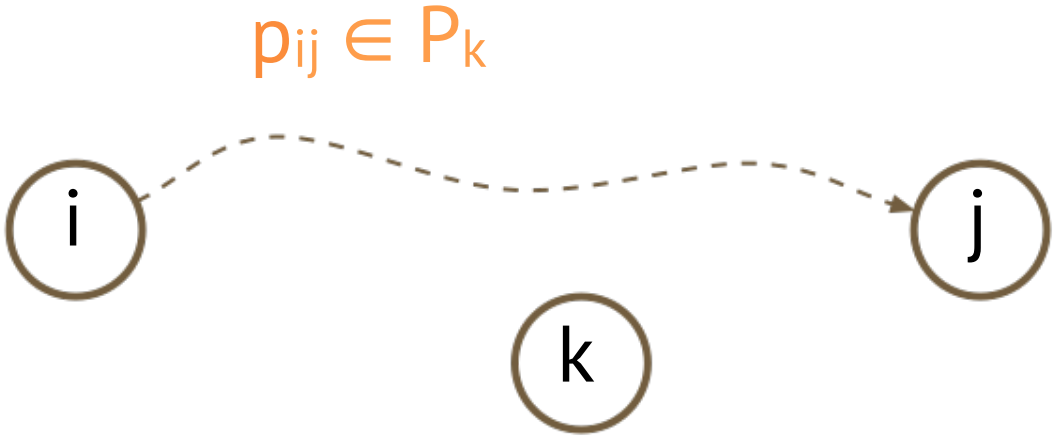
  Falls $k = 0$, so dürfen auf den Pfaden in $P_k=P_0$ überhaupt keine Zwischenknoten liegen, d.h. die Länge der Pfade ist höchstens 1 und wir haben $d_0[i,j] = w(i,j)$.

Floyd-Warshall Algorithmus

- Iteration $k-1 \rightarrow k$

  Es sei p_{ij} ein kürzester Pfad von i nach j in P_k.

  – Fall 1: Ist k kein Zwischenknoten von p_{ij}, so ist p_{ij} auch ein kürzester Pfad von i nach j in P_{k-1} und somit $d_k[i,j] = d_{k-1}[i,j]$

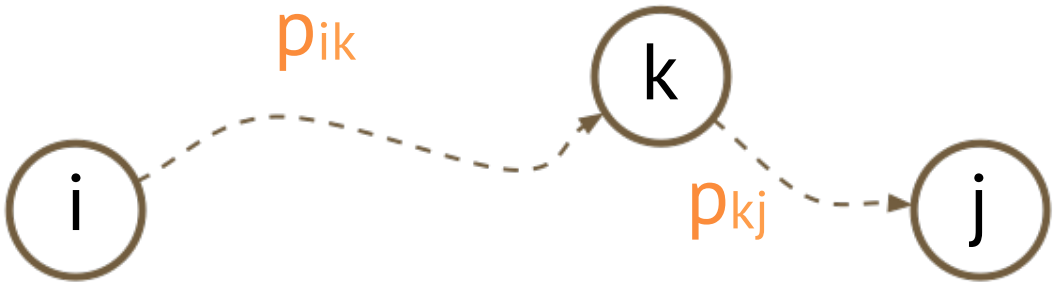


Floyd-Warshall Algorithmus

- Iteration $k-1 \rightarrow k$

  Es sei p_{ij} ein kürzester Pfad von i nach j in P_k.

  – Fall 2: Ist k ein Zwischenknoten von p_{ij}, so ist p_{ik} ein kürzester Pfad von i nach k in P_{k-1} und p_{kj} ein kürzester Pfad von k nach j in P_{k-1} und somit $d_k[i,j] = d_{k-1}[i,k] + d_{k-1}[k,j]$



Floyd-Warshall Algorithmus

- Rekursionsformel

$$d_k[i,j] = \begin{cases} w(i,j) & k = 0 \\ \min(d_{k-1}[i,j],\ d_{k-1}[i,k]+d_{k-1}[k,j]) & k > 0 \end{cases}$$

Floyd-Warshall Algorithmus

- Rekursionsformel für die Vorgänger

  Es sei $p_k[i,j]$ der Vorgänger von j auf einem kürzesten Pfad von i nach j in P_k.

Floyd-Warshall Algorithmus

- Initialisierung k = 0
  Möglich sind nur Pfade der Länge 1.

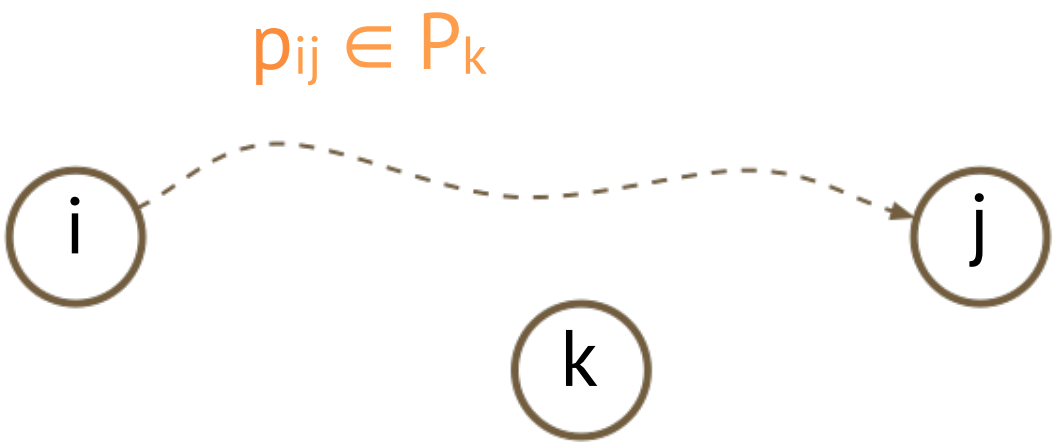
$$p_0[i,j] = \begin{cases} \infty & \text{falls } i = j \text{ oder } w(i,j) = \infty \\ i & \text{falls } i \neq j \text{ und } w(i,j) < \infty \end{cases}$$

Floyd-Warshall Algorithmus

- Iteration $k-1 \rightarrow k$
  - Fall 1: Ein kürzester Pfad in P_k enthält den Knoten k nicht, also:

    $d_{k-1}[i,j] \leqslant d_{k-1}[i,k] + d_{k-1}[k,j]$

    Dann ist $p_k[i,j] = p_{k-1}[i,j]$.

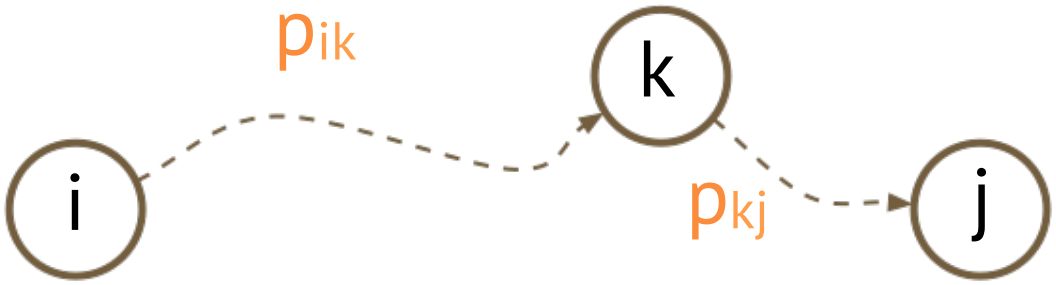


Floyd-Warshall Algorithmus

- Iteration $k-1 \rightarrow k$
  - Fall 2: Ein kürzester Pfad in P_k enthält den Knoten k, also:

    $d_{k-1}[i,j] > d_{k-1}[i,k] + d_{k-1}[k,j]$

    Dann ist $p_k[i,j] = p_{k-1}[k,j]$.



Floyd-Warshall Algorithmus

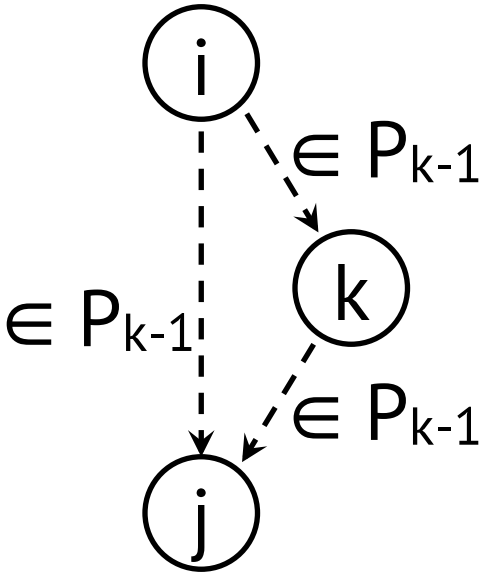
- Rekursionsformel

$$p_k[i,j] = \begin{cases} p_{k-1}[i,j] & \text{falls } d_{k-1}[i,j] \leqslant d_{k-1}[i,k] + d_{k-1}[k,j] \\ p_{k-1}[k,j] & \text{sonst} \end{cases}$$

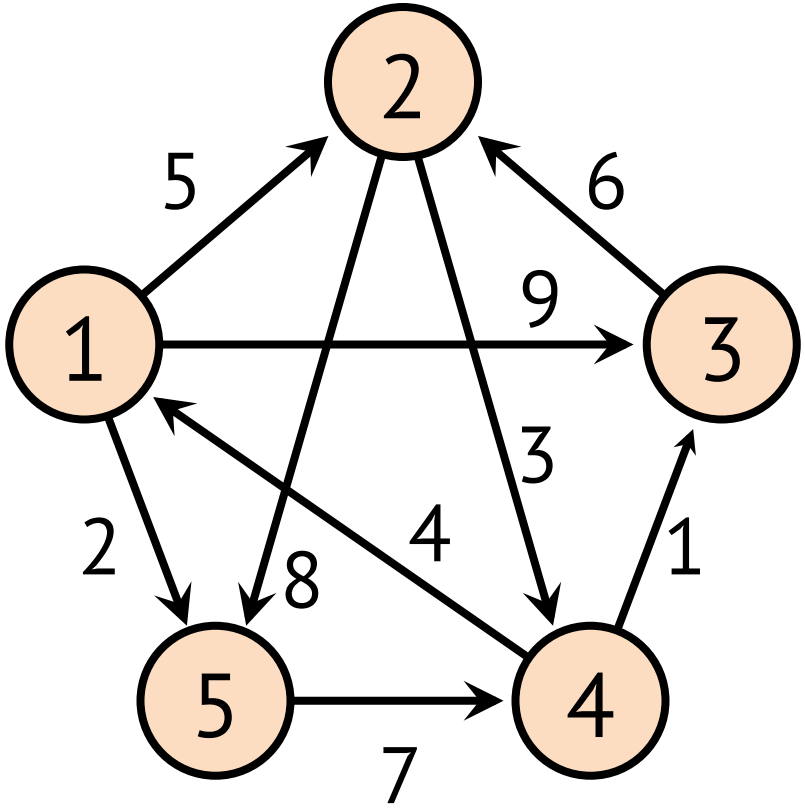
Floyd-Warshall Algorithmus

- FloydWarshall(W)

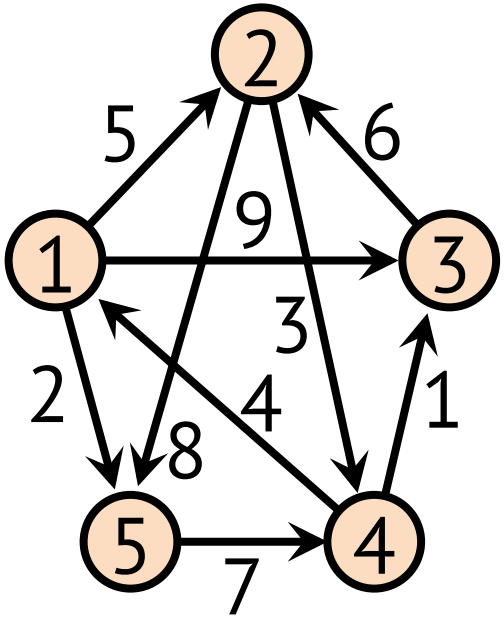
```
D0 ← W
P0 ← …
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if dk−1[i,j] ⩽ dk−1[i,k]+dk−1[k,j] then
                dk[i,j] ← dk−1[i,j]
                pk[i,j] ← pk−1[i,j]
            else
                dk[i,j] ← dk−1[i,k]+dk−1[k,j]
                pk[i,j] ← pk−1[k,j]
return Dn, Pn
```



Beispiel



Beispiel



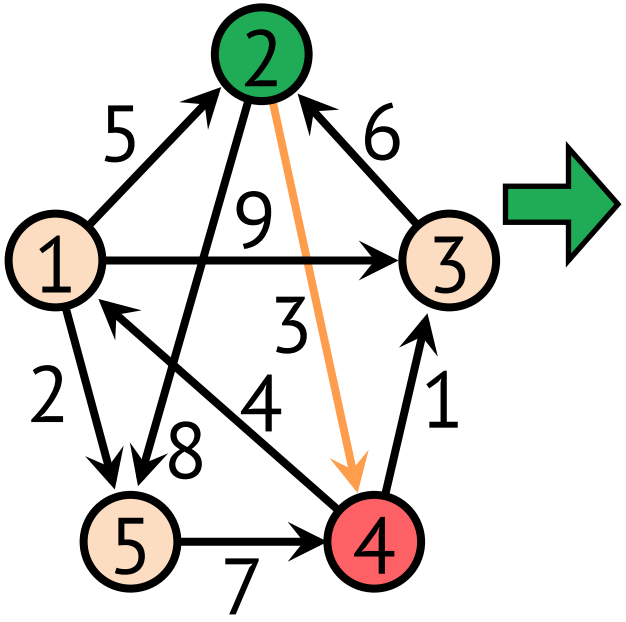
$D_0 =$

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	∞	1	0	∞
∞	∞	∞	7	0

$P_0 =$

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	∞	4	∞	∞
∞	∞	∞	5	∞

Beispiel



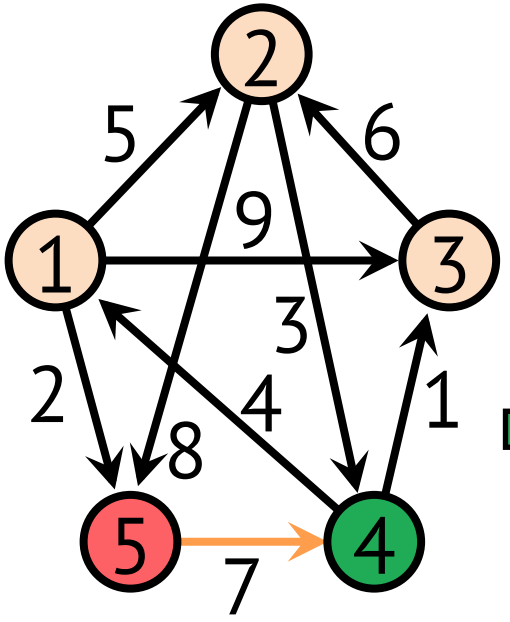
$D_0 =$

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	∞	1	0	∞
∞	∞	∞	7	0

$P_0 =$

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	∞	4	∞	∞
∞	∞	∞	5	∞

Beispiel



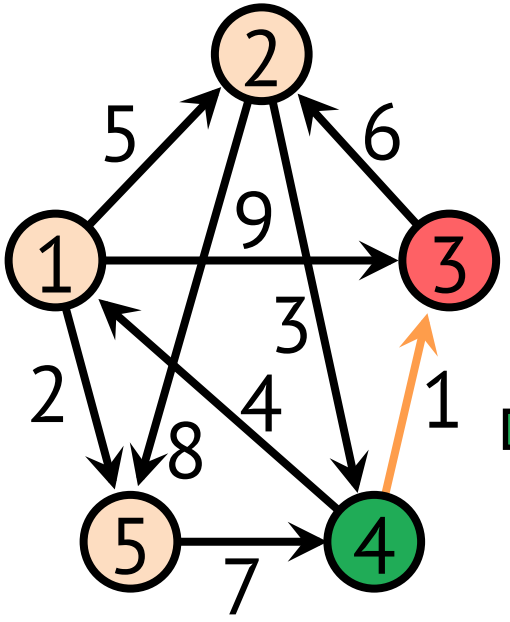
$D_0 =$

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	∞	1	0	∞
∞	∞	∞	7	0

$P_0 =$

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	∞	4	∞	∞
∞	∞	∞	5	∞

Beispiel



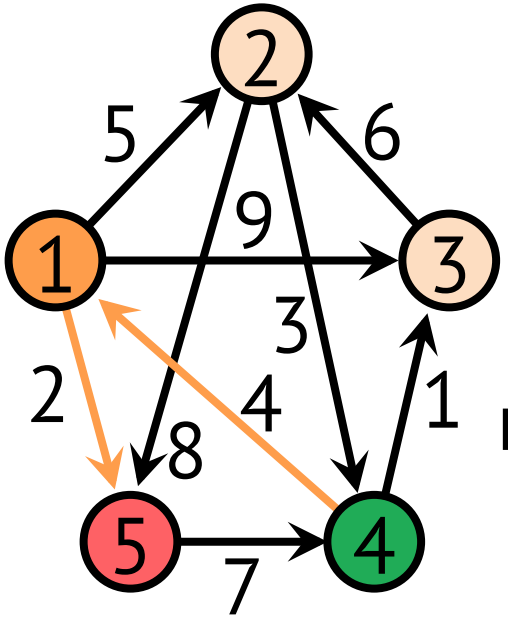
$D_{0\to 1}$ =

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	∞	1	0	∞
∞	∞	∞	7	0

$P_{0\to 1}$ =

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	∞	4	∞	∞
∞	∞	∞	5	∞

Beispiel



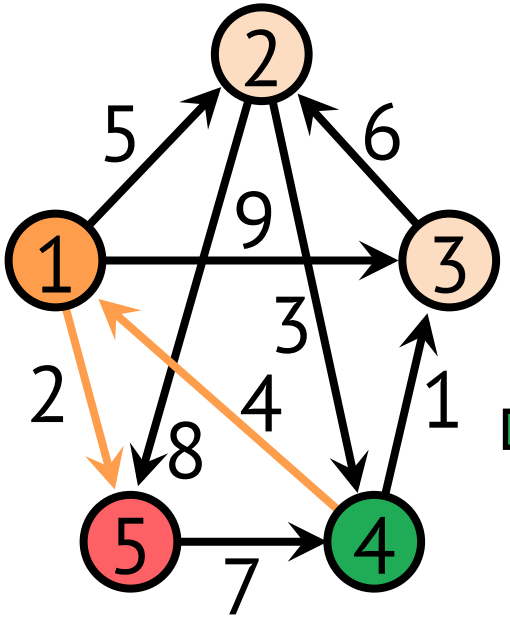
$D_{0\to1}$ =

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	∞	1	0	∞
∞	∞	∞	7	0

$P_{0\to1}$ =

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	∞	4	∞	∞
∞	∞	∞	5	∞

Beispiel



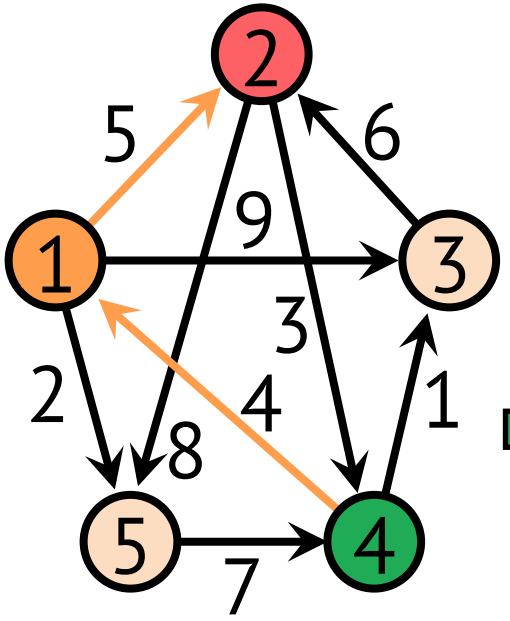
$D_{0\to 1}$ =

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	∞	1	0	6
∞	∞	∞	7	0

$P_{0\to 1}$ =

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	∞	4	∞	1
∞	∞	∞	5	∞

Beispiel



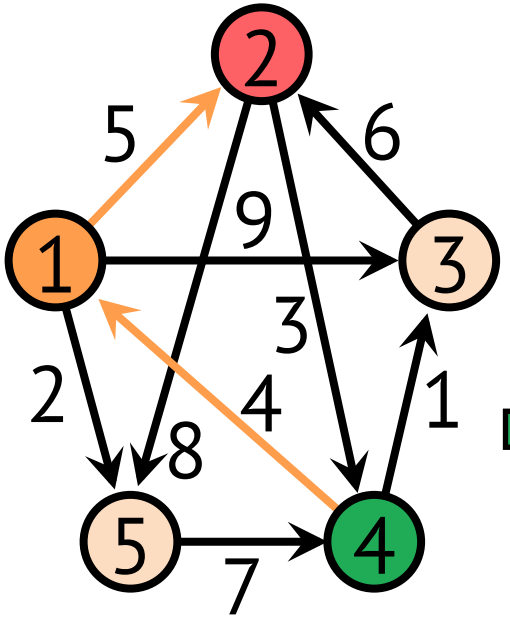
$D_{0 \to 1} =$

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	∞	1	0	6
∞	∞	∞	7	0

$P_{0 \to 1} =$

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	∞	4	∞	1
∞	∞	∞	5	∞

Beispiel



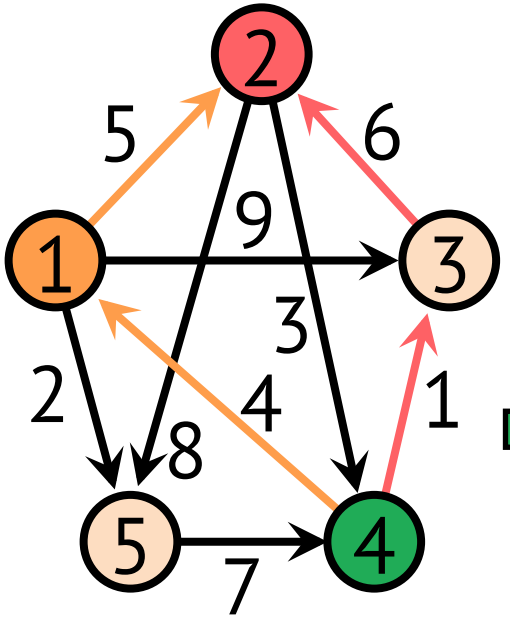
$D_{0 \to 1} =$

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	9	1	0	6
∞	∞	∞	7	0

$P_{0 \to 1} =$

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



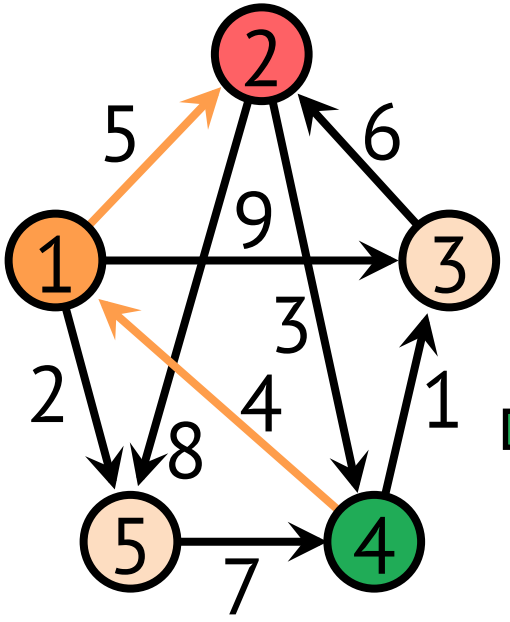
$D_{0 \to 1}$ =

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	9	1	0	6
∞	∞	∞	7	0

$P_{0 \to 1}$ =

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



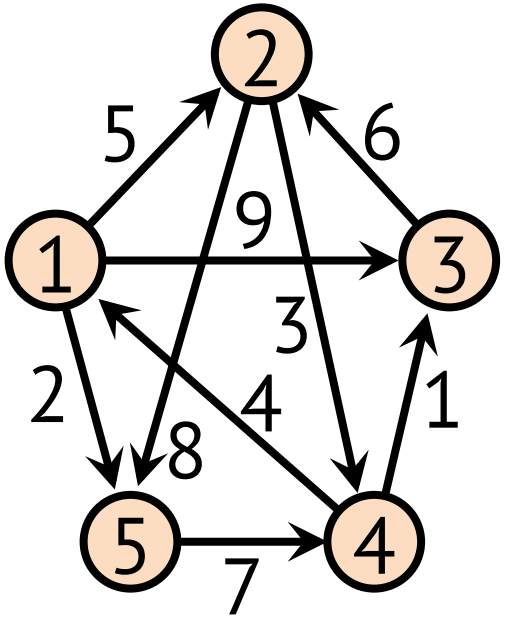
$D_{0 \to 1} =$

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	9	1	0	6
∞	∞	∞	7	0

$P_{0 \to 1} =$

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



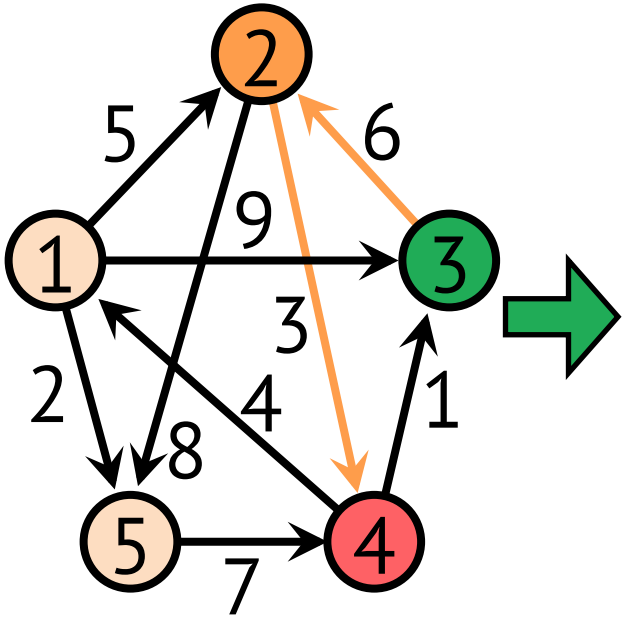
$D_1 =$

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	9	1	0	6
∞	∞	∞	7	0

$P_1 =$

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



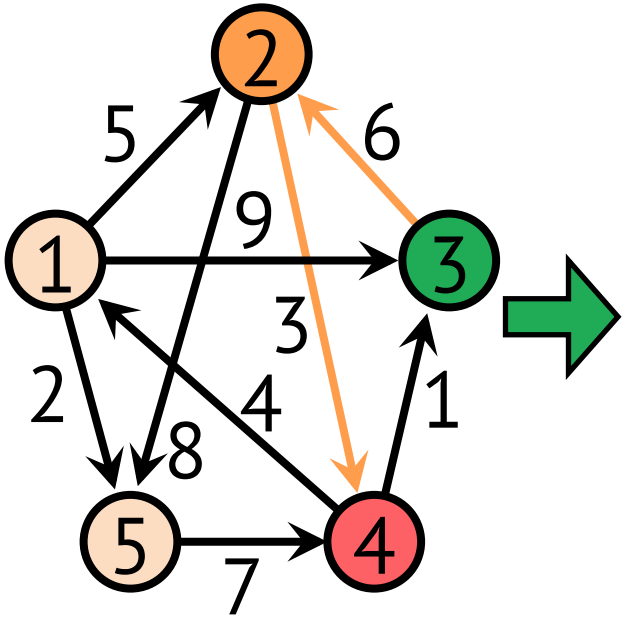
$D_{1\to 2}$ =

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	∞	∞
4	9	1	0	6
∞	∞	∞	7	0

$P_{1\to 2}$ =

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	∞	∞
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



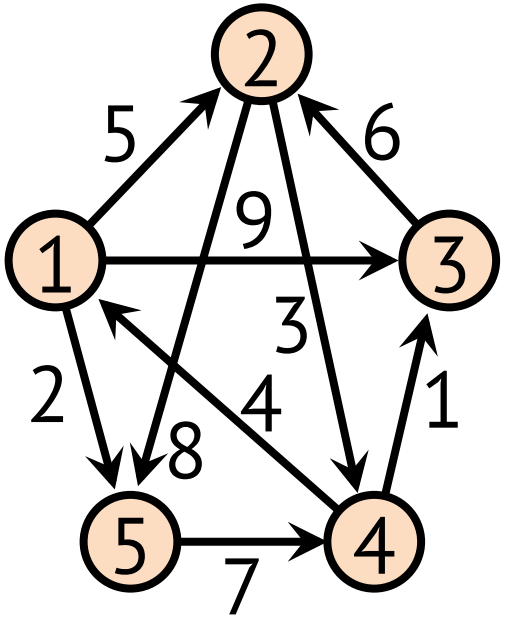
$D_{1\to 2}$ =

0	5	9	∞	2
∞	0	∞	3	8
∞	6	0	9	∞
4	9	1	0	6
∞	∞	∞	7	0

$P_{1\to 2}$ =

∞	1	1	∞	1
∞	∞	∞	2	2
∞	3	∞	2	∞
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



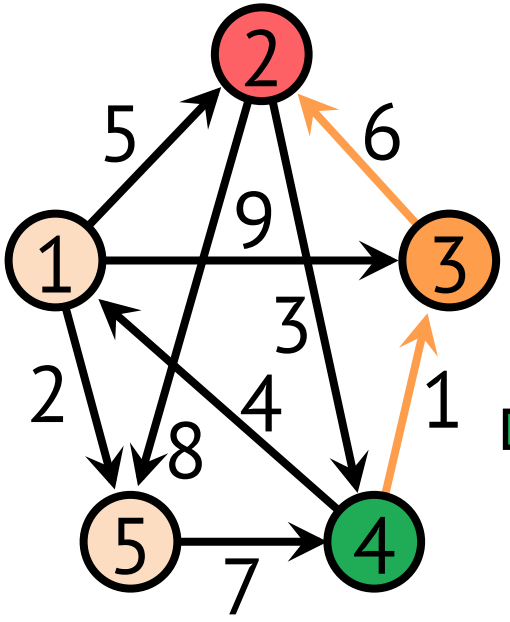
$D_2 =$

0	5	9	8	2
∞	0	∞	3	8
∞	6	0	9	14
4	9	1	0	6
∞	∞	∞	7	0

$P_2 =$

∞	1	1	2	1
∞	∞	∞	2	2
∞	3	∞	2	2
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



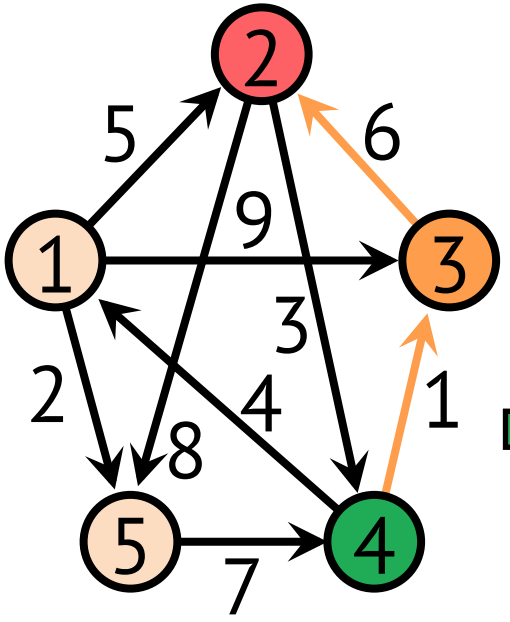
$D_{2\to 3}$

0	5	9	8	2
∞	0	∞	3	8
∞	6	0	9	14
4	9	1	0	6
∞	∞	∞	7	0

$P_{2\to 3}$

∞	1	1	2	1
∞	∞	∞	2	2
∞	3	∞	2	2
4	1	4	∞	1
∞	∞	∞	5	∞

Beispiel



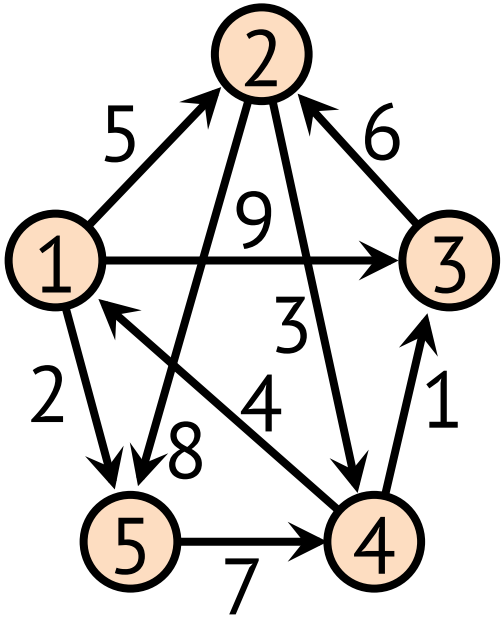
$D_{2\to 3}$

0	5	9	8	2
∞	0	∞	3	8
∞	6	0	9	14
4	7	1	0	6
∞	∞	∞	7	0

$P_{2\to 3}$

∞	1	1	2	1
∞	∞	∞	2	2
∞	3	∞	2	2
4	3	4	∞	1
∞	∞	∞	5	∞

Beispiel



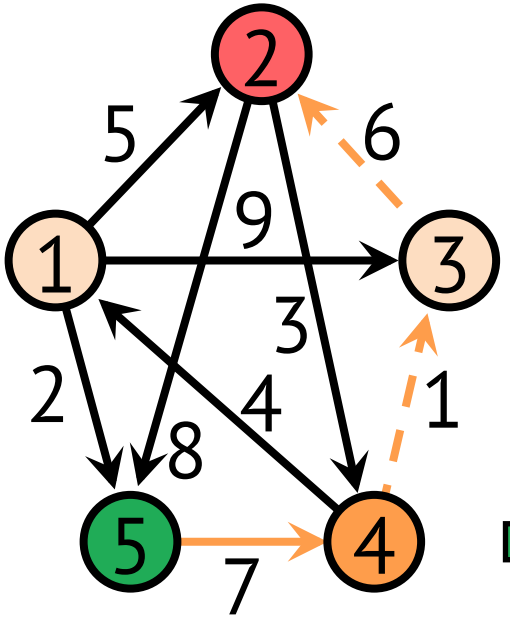
$D_3 =$

0	5	9	8	2
∞	0	∞	3	8
∞	6	0	9	14
4	7	1	0	6
∞	∞	∞	7	0

$P_3 =$

∞	1	1	2	1
∞	∞	∞	2	2
∞	3	∞	2	2
4	3	4	∞	1
∞	∞	∞	5	∞

Beispiel



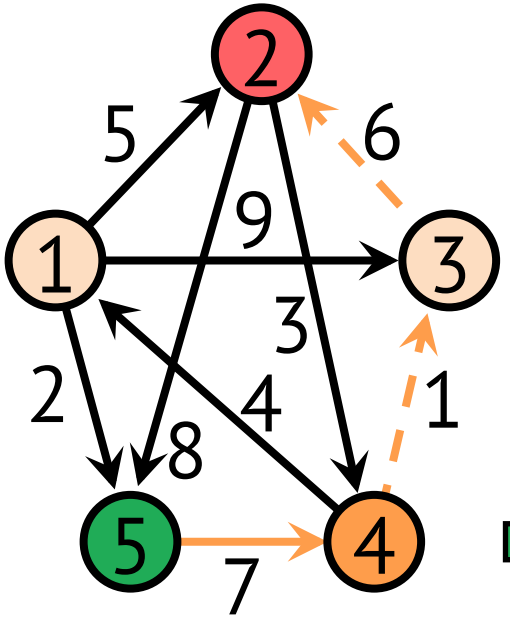
$D_{3\to 4}$ =

0	5	9	8	2
∞	0	∞	3	8
∞	6	0	9	14
4	7	1	0	6
∞	∞	∞	7	0

$P_{3\to 4}$ =

∞	1	1	2	1
∞	∞	∞	2	2
∞	3	∞	2	2
4	3	4	∞	1
∞	∞	∞	5	∞

Beispiel



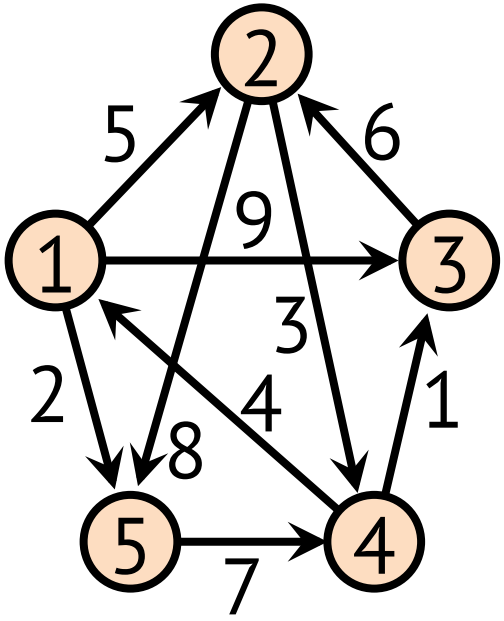
$D_{3\to 4}$ =

0	5	9	8	2
∞	0	∞	3	8
∞	6	0	9	14
4	7	1	0	6
∞	14	∞	7	0

$P_{3\to 4}$ =

∞	1	1	2	1
∞	∞	∞	2	2
∞	3	∞	2	2
4	3	4	∞	1
∞	3	∞	5	∞

Beispiel



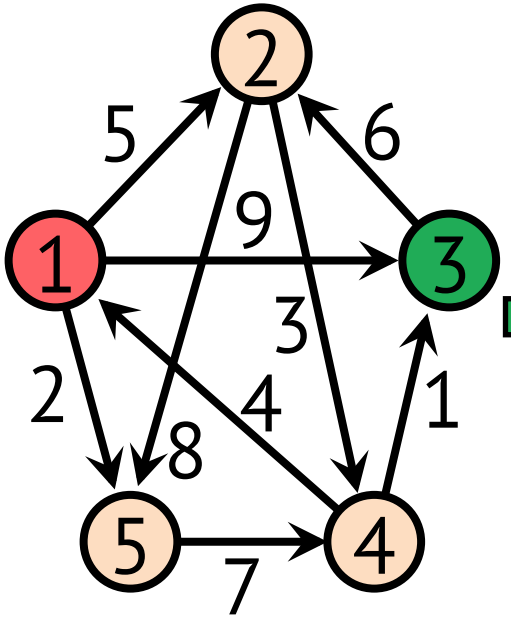
$D_5 =$

0	5	9	8	2
7	0	4	3	8
13	6	0	9	14
4	7	1	0	6
11	14	8	7	0

$P_5 =$

∞	1	1	2	1
4	∞	4	2	2
4	3	∞	2	2
4	3	4	∞	1
4	3	4	5	∞

Beispiel



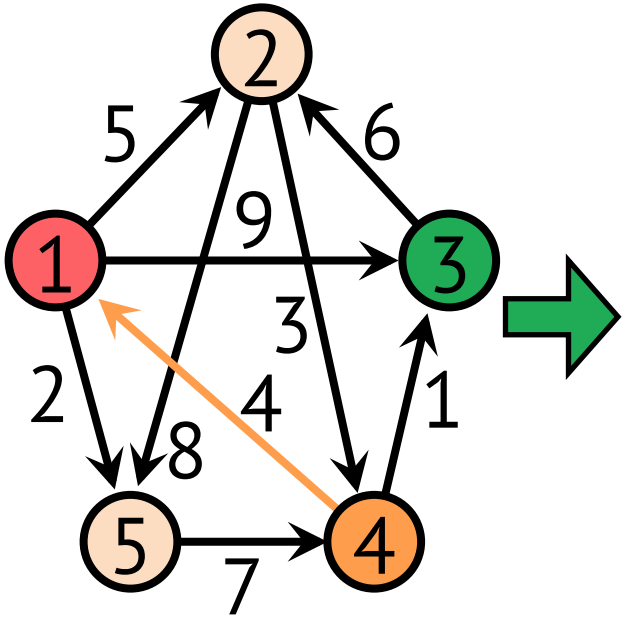
$D_1 =$

0	5	9	8	2
7	0	4	3	8
13	6	0	9	14
4	7	1	0	6
11	14	8	7	0

$P_1 =$

∞	1	1	2	1
4	∞	4	2	2
4	3	∞	2	2
4	3	4	∞	1
4	3	4	5	∞

Beispiel



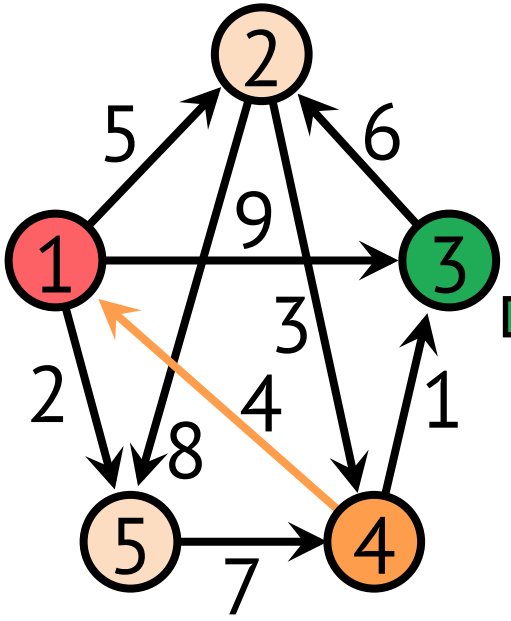
$D_4 =$

0	5	9	8	2
7	0	4	3	8
13	6	0	9	14
4	7	1	0	6
11	14	8	7	0

$P_4 =$

∞	1	1	2	1
4	∞	4	2	2
4	3	∞	2	2
4	3	4	∞	1
4	3	4	5	∞

Beispiel



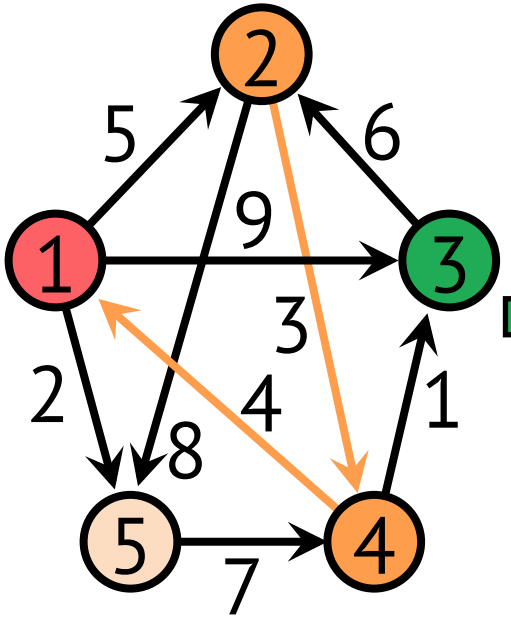
$D_4 =$

0	5	9	8	2
7	0	4	3	8
13	6	0	9	14
4	7	1	0	6
11	14	8	7	0

$P_5 =$

∞	1	1	2	1
4	∞	4	2	2
4	3	∞	2	2
4	3	4	∞	1
4	3	4	5	∞

Beispiel



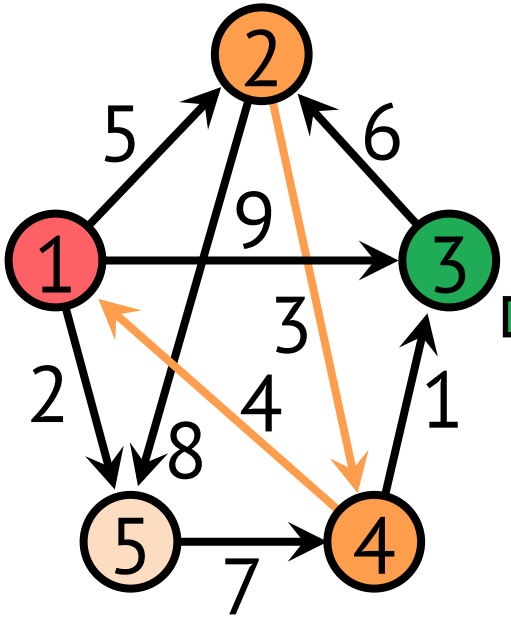
$D_4 =$

0	5	9	8	2
7	0	4	3	8
13	6	0	9	14
4	7	1	0	6
11	14	8	7	0

$P_5 =$

∞	1	1	2	1
4	∞	4	2	2
4	3	∞	2	2
4	3	4	∞	1
4	3	4	5	∞

Beispiel



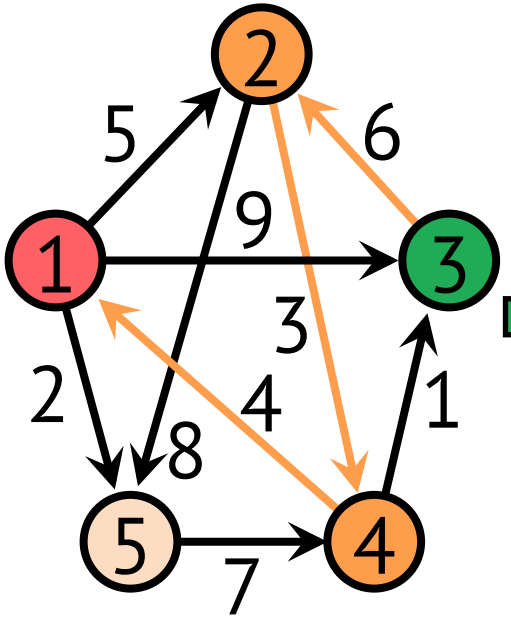
$D_4 =$

0	5	9	8	2
7	0	4	3	8
13	6	0	9	14
4	7	1	0	6
11	14	8	7	0

$P_5 =$

∞	1	1	2	1
4	∞	4	2	2
4	3	∞	2	2
4	3	4	∞	1
4	3	4	5	∞

Beispiel



$D_4 =$

0	5	9	8	2
7	0	4	3	8
13	6	0	9	14
4	7	1	0	6
11	14	8	7	0

$P_5 =$

∞	1	1	2	1
4	∞	4	2	2
4	3	∞	2	2
4	3	4	∞	1
4	3	4	5	∞

Floyd-Warshall Algorithmus

- FloydWarshall(W)

$O(n^2)$
$D_0 \leftarrow W$
$P_0 \leftarrow \ldots$

$O(n^3)$
for k ← 1 to n do
  for i ← 1 to n do
    for j ← 1 to n do
      $d_{k,ij} \leftarrow \min(d_{k-1,ij}, d_{k-1,ik}+d_{k-1,kj})$
      $p_{k,ij} \leftarrow \ldots$
return D_n

Floyd-Warshall Algorithmus

- Fazit

  Der Floyd-Warshall Algorithmus hat einen Aufwand von

  $$O(V^3)$$

2.6.4 Kürzeste Pfade

Transitive Hülle

- Transitive Hülle

  Es sei G=(V,E) mit V={1,...,n} ein Graph.

  Die transitive Hülle von G ist definiert als

  $$G^*=(V,E^*)$$

  mit

  E^* = { (i,j): es ex. Pfad von i nach j in G }

Transitive Hülle

- Lösung

  Ordne jeder Kante in E das Gewicht 1 zu und führe den Floyd-Warshall Algorithmus aus. Gibt es einen Weg von i nach j, so erhalten wir $d[i,j] < n$, andernfalls $d[i,j] = \infty$.

Transitive Hülle

- In der Praxis schneller

  Ersetze min und + im Floyd-Warshall Algorithmus durch die logischen Operatoren $\vee$ (oder) und $\wedge$ (und).

Transitive Hülle

- Definition

  Es sei $t_k[i,j] = 1$ falls ein Pfad von i nach j in P_k existiert. Insbesondere ist also $t_n[i,j] = 1$, falls $(i,j) \in E^*$.

Transitive Hülle

- Rekursionsformel

$$t_0[i,j] = \begin{cases} 0 & \text{falls } i \neq j \text{ und } (i,j) \notin E \\ 1 & \text{sonst} \end{cases}$$

$$t_k[i,j] = t_{k-1}[i,j] \vee (t_{k-1}[i,k] \wedge t_{k-1}[k,j])$$

Transitive Hülle

- TransitiveClosure(G)

```
for i ← 1 to n do
    for j ← 1 to n do
        t0[i,j] ← (i = j) ∨ (i,j) ∈ E
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            tk[i,j] = tk−1[i,j] ∨ (tk−1[i,k] ∧ tk−1[k,j])
return Tn
```

Transitive Hülle

- Fazit

  Der Algorithmus zur Berechnung der transitiven Hülle hat den gleichen Aufwand, wie der Floyd-Warshall Algorithmus, nämlich

  $$O(V^3)$$

  In der Praxis können jedoch logische Operatoren schneller bearbeitet werden als arithmetische Operatoren.