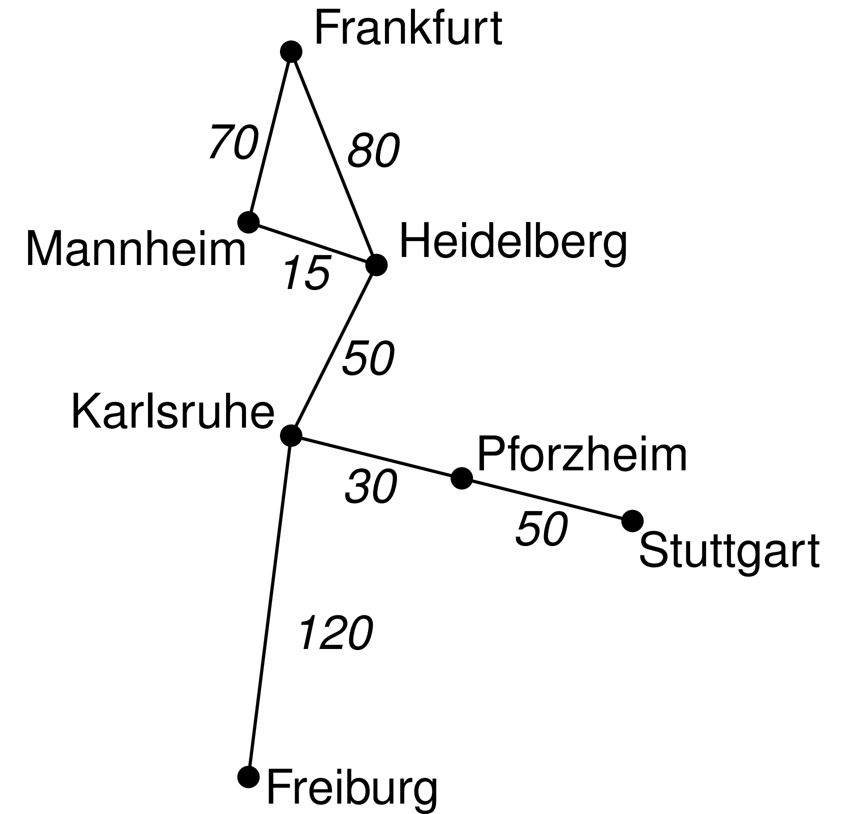
Karlsruher Institut für Technologie

Algorithmen 1

Übung 4
Graphen

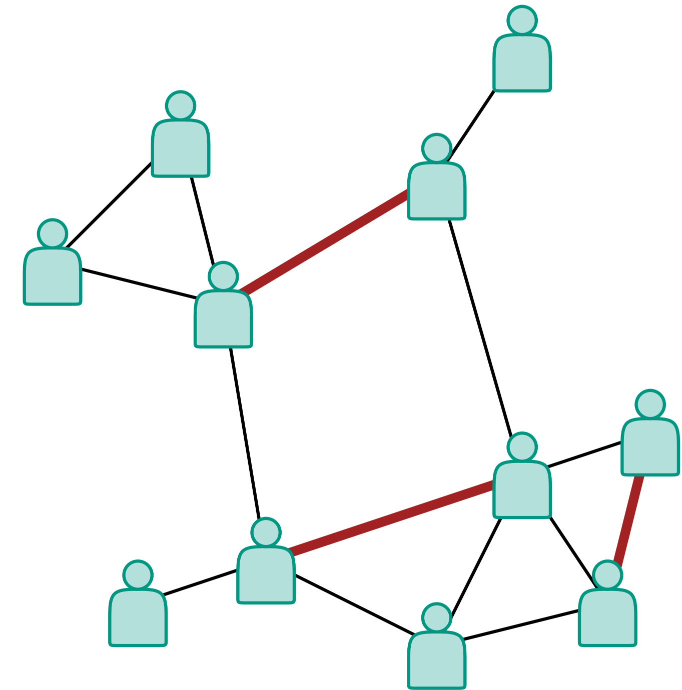
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke



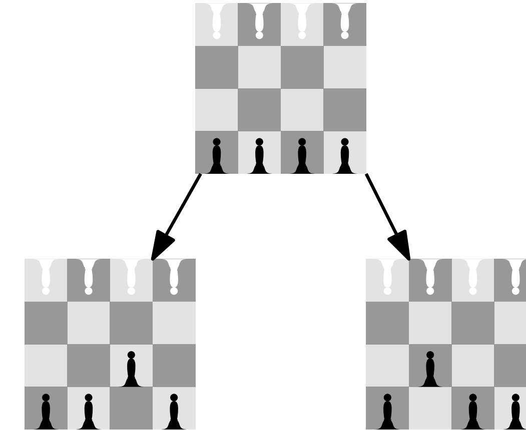
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke



Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke
  - sonstiges
- viele praktische Fragestellungen sind algorithmische Probleme auf Graphen



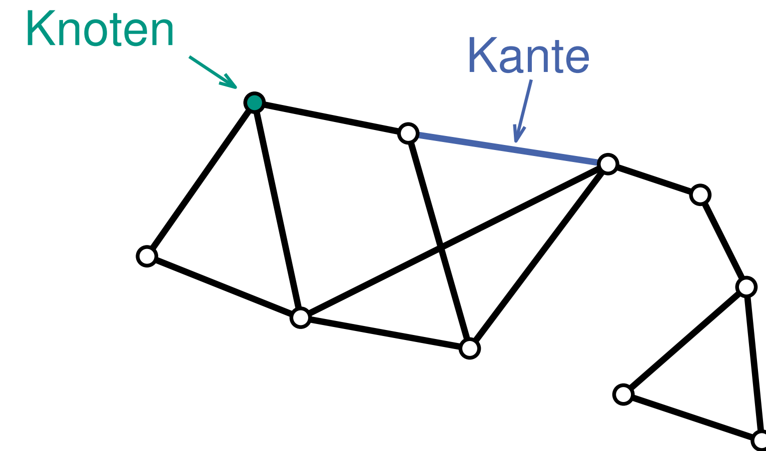
Grundlagen und Notation

Karlsruher Institut für Technologie

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



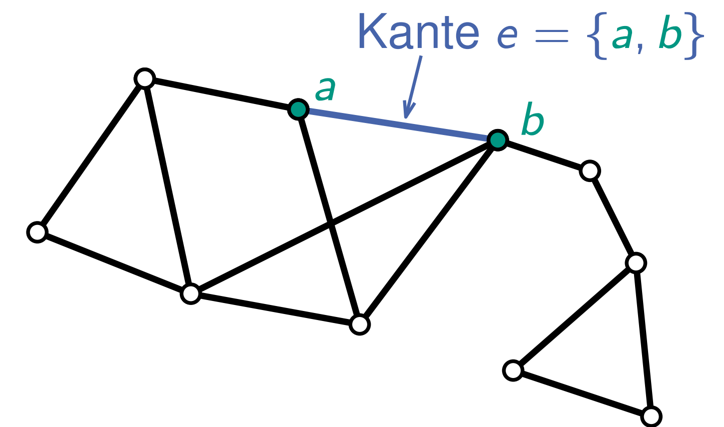
Grundlagen und Notation

Karlsruher Institut für Technologie

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

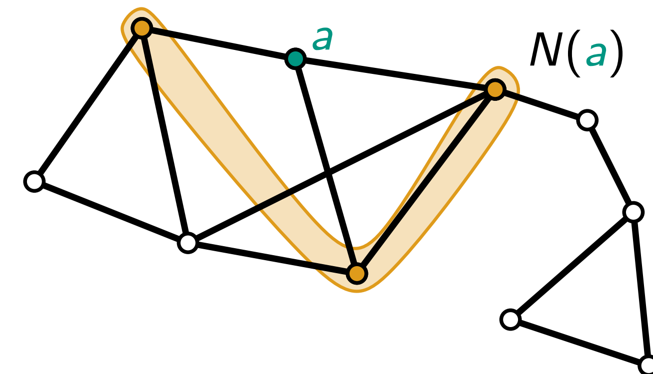


Grundlagen und Notation

Graph

$G = (V, E)$

- E: Teilmenge von $\binom{V}{2}$
- V: endliche Menge



Relationen

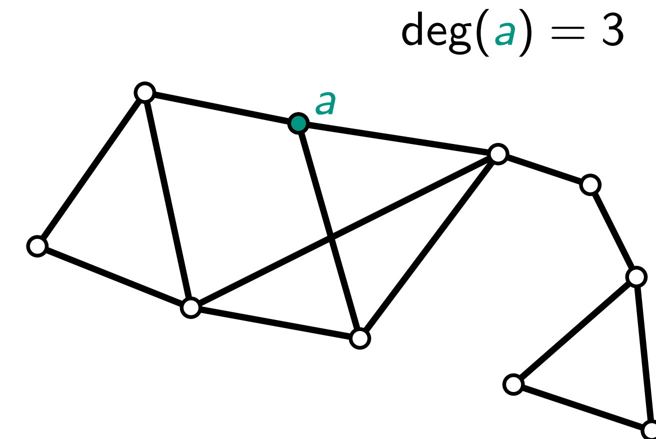
- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



Relationen

- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

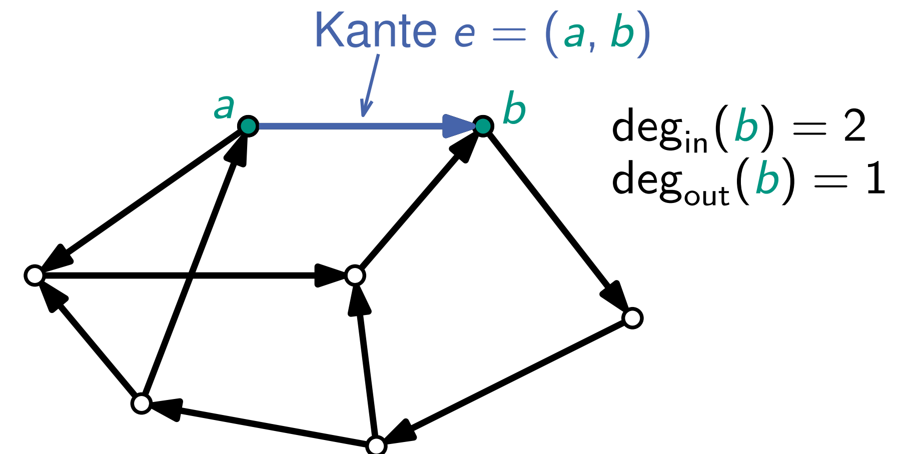
$G = (V, E)$

- E: Teilmenge von $\binom{V}{2}$
- V: endliche Menge

Graph (gerichtet)

$G = (V, E)$

- E: Teilmenge von $V \times V$
- V: endliche Menge



Grundlagen und Notation

Karlsruher Institut für Technologie

Graph

$G = (V, E)$

- E: Teilmenge von $\binom{V}{2}$
- V: endliche Menge

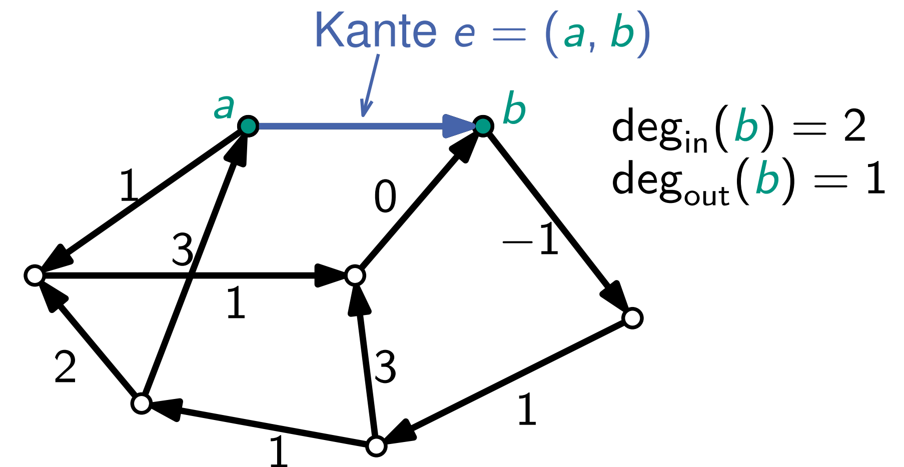
Graph (gerichtet)

$G = (V, E)$

- E: Teilmenge von $V \times V$
- V: endliche Menge

Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$



Grundlagen und Notation

Karlsruher Institut für Technologie

Graph

$G = (V, E)$

- E: Teilmenge von $\binom{V}{2}$
- V: endliche Menge

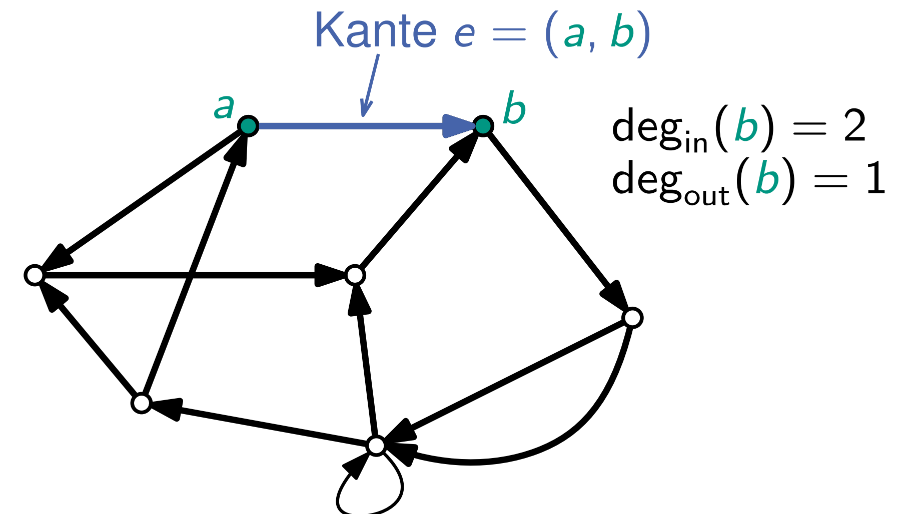
Graph (gerichtet)

$G = (V, E)$

- E: Teilmenge von $V \times V$
- V: endliche Menge

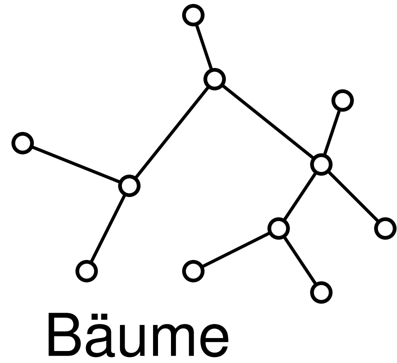
Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$
- Multigraphen



Besondere Graphen

Karlsruher Institut für Technologie



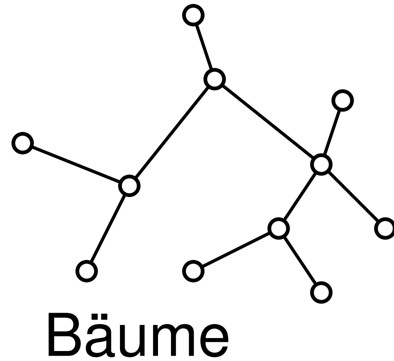
Bäume

Charakterisierung:

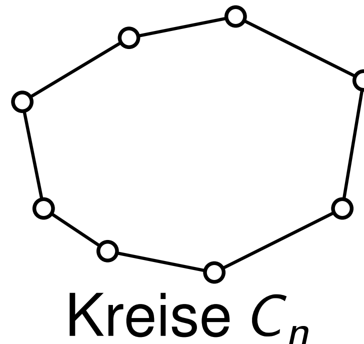
- kreisfrei
- zusammenhängend
- $m = n - 1$

Zwei Eigenschaften implizieren dritte

Besondere Graphen



Bäume

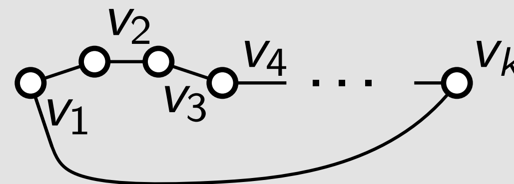


Kreise C_n

Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

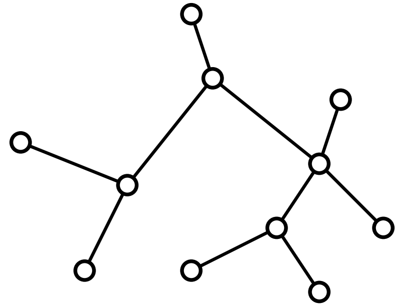
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



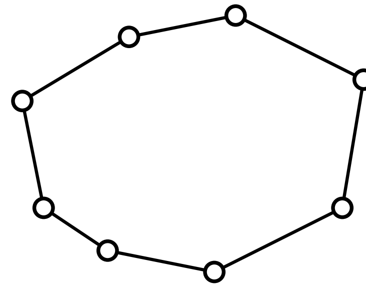
- für $2 \leq i < k - 1$: $\{v_i, v_k\} \notin E$
- $\{v_k, v_1\} \in E$ falls $k = n$

Besondere Graphen

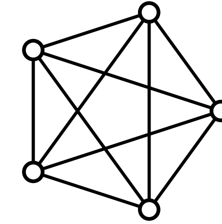
Karlsruher Institut für Technologie



Bäume



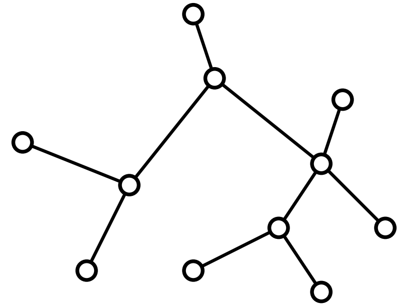
Kreise



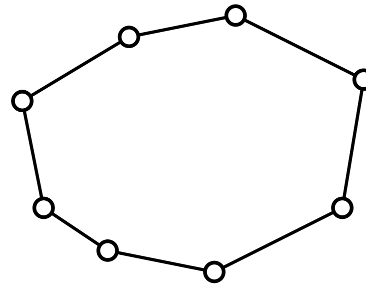
$m = \binom{n}{2} = \frac{n(n-1)}{2}$

vollst. Graph K_n

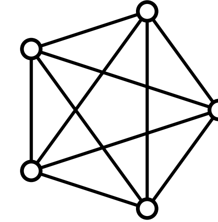
Besondere Graphen



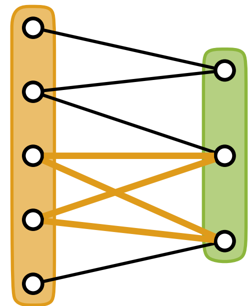
Bäume



Kreise



vollst. Graph K_n

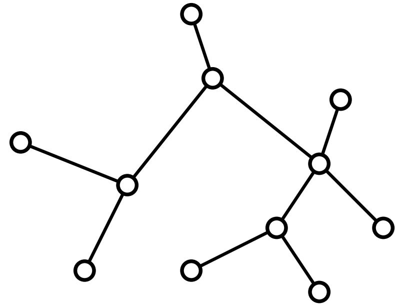


bipartiter Graph

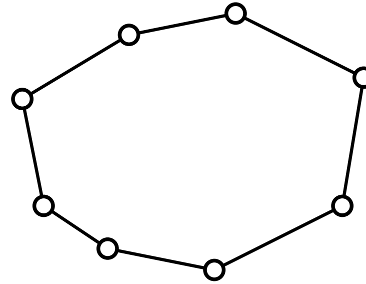
$G = (A \cup B, E)$

Charakterisierung:
G enthält nur Kreise *gerader* Länge in G

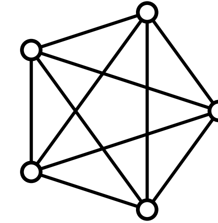
Besondere Graphen



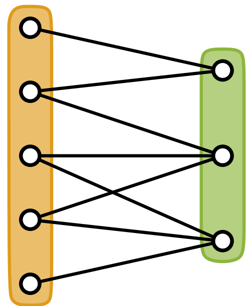
Bäume



Kreise

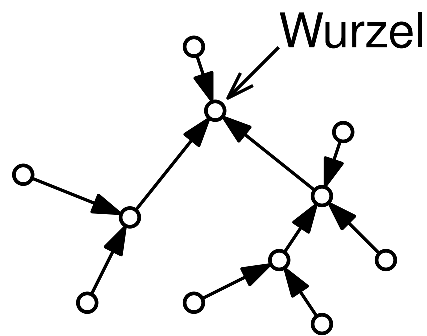


vollst. Graph K_n



bipartiter Graph

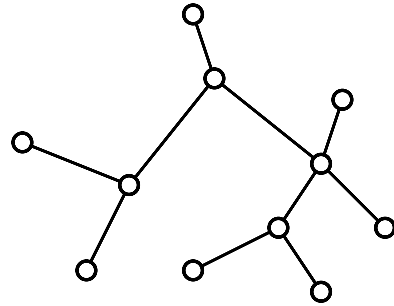
$G = (A \cup B, E)$



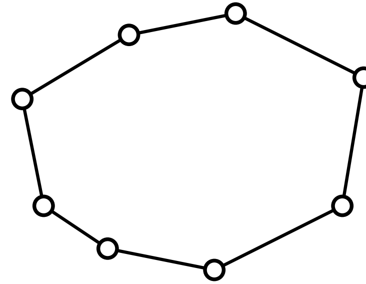
In-tree / zur Wurzel ger. Baum

Besondere Graphen

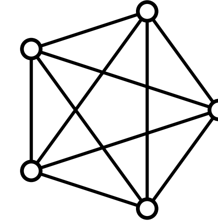
Karlsruher Institut für Technologie



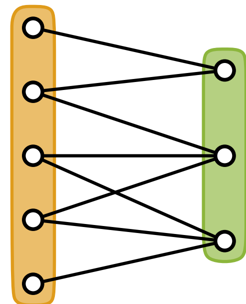
Bäume



Kreise

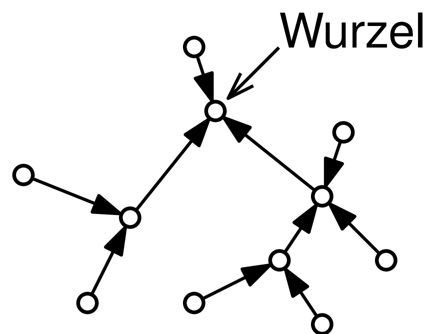


vollst. Graph K_n

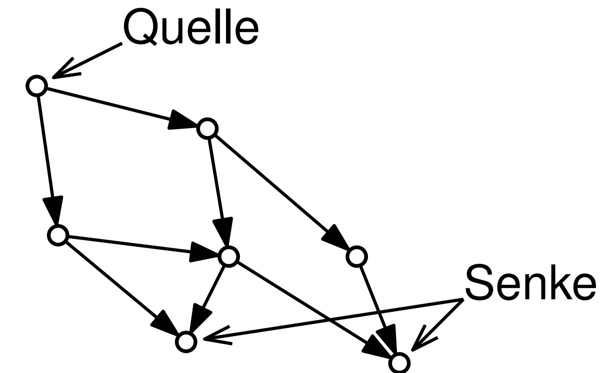


bipartiter Graph

$G = (A \cup B, E)$



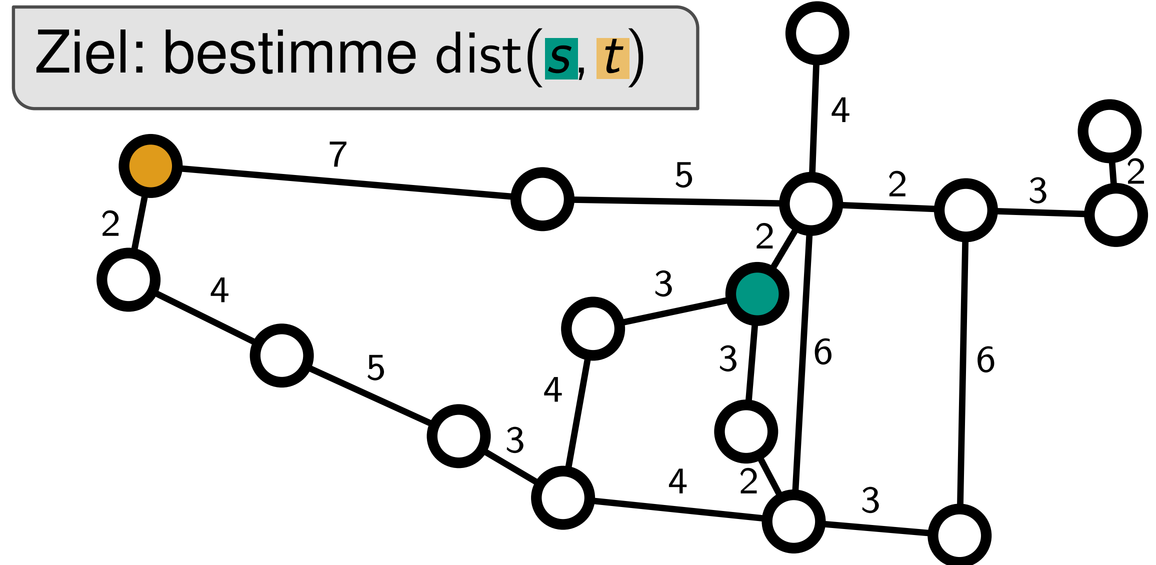
In-tree / zur Wurzel ger. Baum



directed acyclic graph (DAG)

Dijkstra (Wiederholung)

```
Dijkstra(Graph G, Node s)
  d := Array of size n initialized with ∞
  d[s] := 0
  PriorityQueue Q := empty priority queue
  for Node v in V do
    Q.push(v, d[v])
  while Q ≠ ∅ do
    u := Q.popMin()
    for Node v in N(u) do
      if d[v] > d[u] + len(u, v) then
        d[v] := d[u] + len(u, v)
        Q.decPrio(v, d[v])
```



Dijkstra (Wiederholung)

Dijkstra(*Graph* G, *Node* s)

$d := $ *Array* of size n initialized with ∞

$d[s] := 0$

PriorityQueue $Q :=$ empty priority queue

for *Node* v in V **do**

  $Q.$**push**$(v, d[v])$

while $Q \neq \emptyset$ **do**

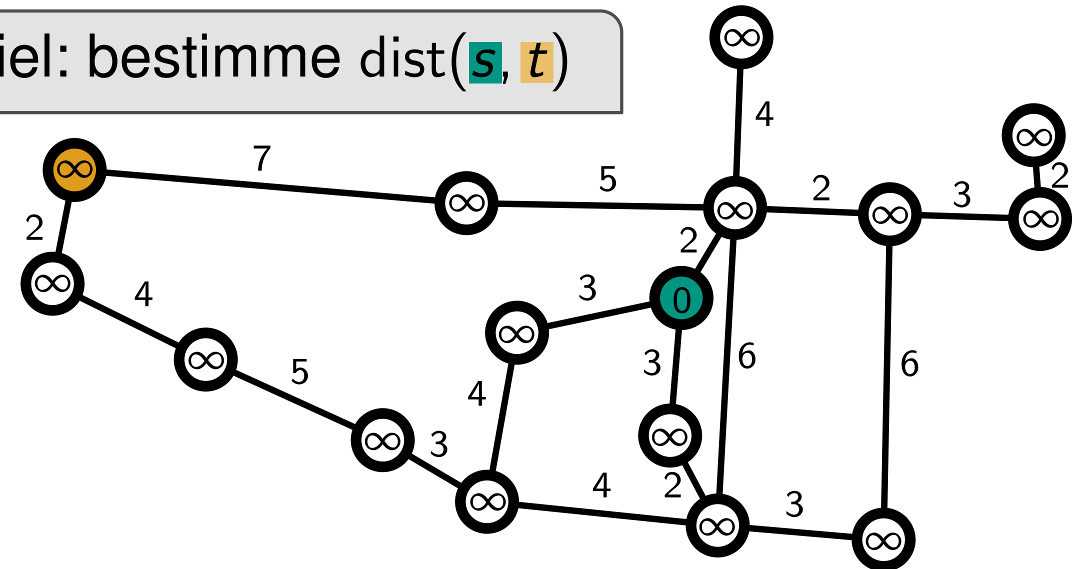
  $u := Q.$**popMin**$()$

  for *Node* v in $N(u)$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

      $Q.$**decPrio**$(v, d[v])$



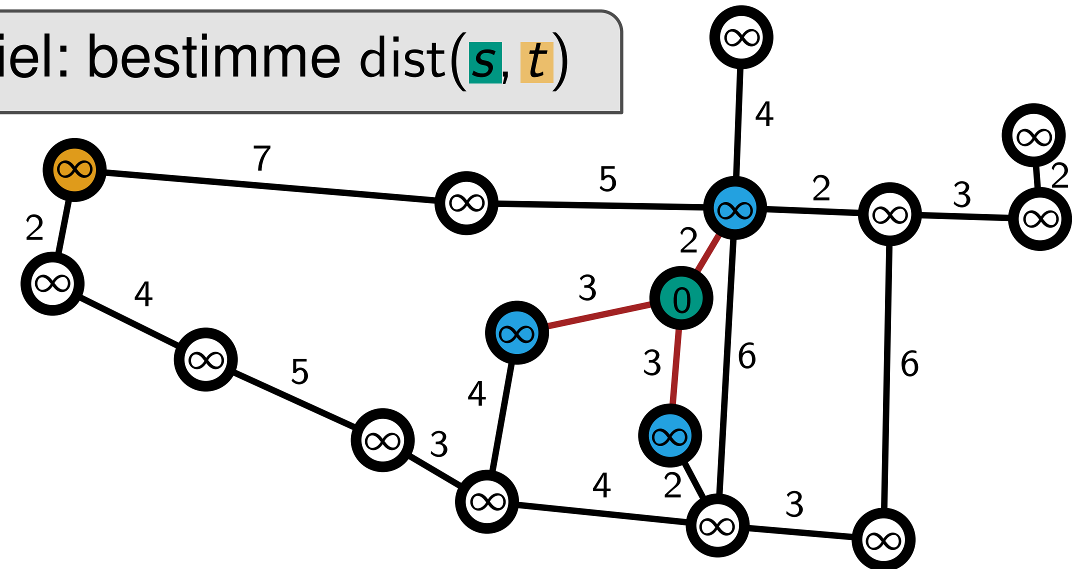
Dijkstra (Wiederholung)

Karlsruher Institut für Technologie

Dijkstra(*Graph* G, *Node* s)

- $d :=$ *Array* of size n initialized with ∞
- $d[s] := 0$
- *PriorityQueue* $Q :=$ empty priority queue
- **for** *Node* v in V **do**
  - Q.**push**$(v, d[v])$
- **while** $Q \neq \emptyset$ **do**
  - $u := Q$.**popMin**()
  - **for** *Node* v in $N(u)$ **do**
    - **if** $d[v] > d[u] + \text{len}(u, v)$ **then**
      - $d[v] := d[u] + \text{len}(u, v)$
      - Q.**decPrio**$(v, d[v])$

Ziel: bestimme dist(s, t)



Dijkstra (Wiederholung)

Karlsruher Institut für Technologie

Dijkstra(*Graph G*, *Node s*)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

PriorityQueue $Q :=$ empty priority queue

for *Node* v in V **do**

  Q.**push**$(v, d[v])$

while $Q \neq \emptyset$ **do**

  $u := Q$.**popMin**()

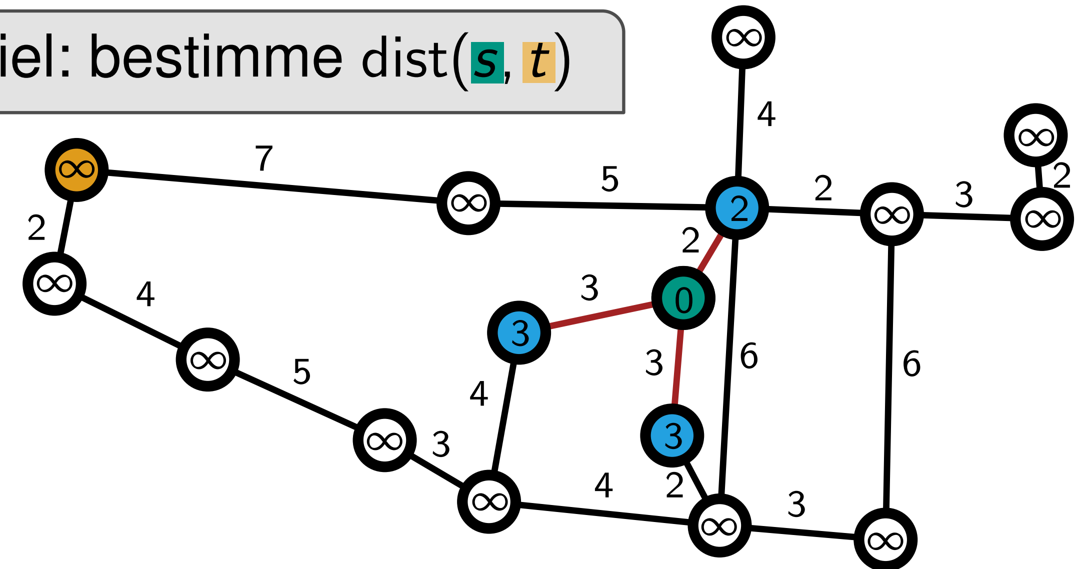
  for *Node* v in $N(u)$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

      Q.**decPrio**$(v, d[v])$

Ziel: bestimme dist(s, t)

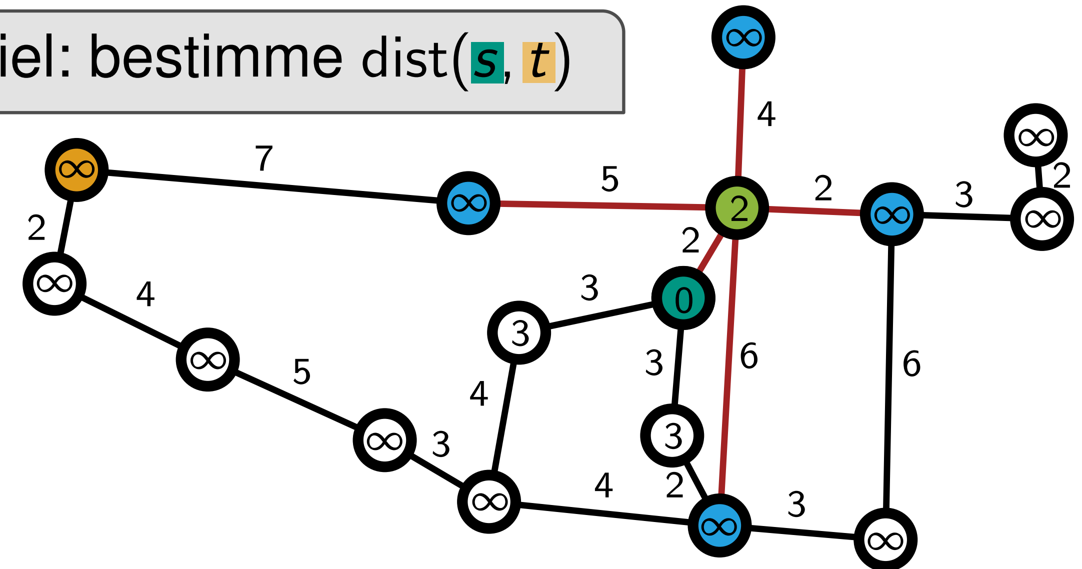


Dijkstra (Wiederholung)

Karlsruher Institut für Technologie

Dijkstra(*Graph G*, *Node s*)

- $d :=$ *Array* of size n initialized with ∞
- $d[s] := 0$
- *PriorityQueue* $Q :=$ empty priority queue
- **for** *Node* v in V **do**
  - Q.**push**$(v, d[v])$
- **while** $Q \neq \emptyset$ **do**
  - $u := Q$.**popMin**()
  - **for** *Node* v in $N(u)$ **do**
    - **if** $d[v] > d[u] + \text{len}(u, v)$ **then**
      - $d[v] := d[u] + \text{len}(u, v)$
      - Q.**decPrio**$(v, d[v])$



Dijkstra (Wiederholung)

Dijkstra(*Graph* G, *Node* s)

d := *Array* of size n initialized with ∞

$d[s] := 0$

PriorityQueue Q := empty priority queue

for *Node* v in V **do**

  Q.**push**(v, $d[v]$)

while $Q \neq \emptyset$ **do**

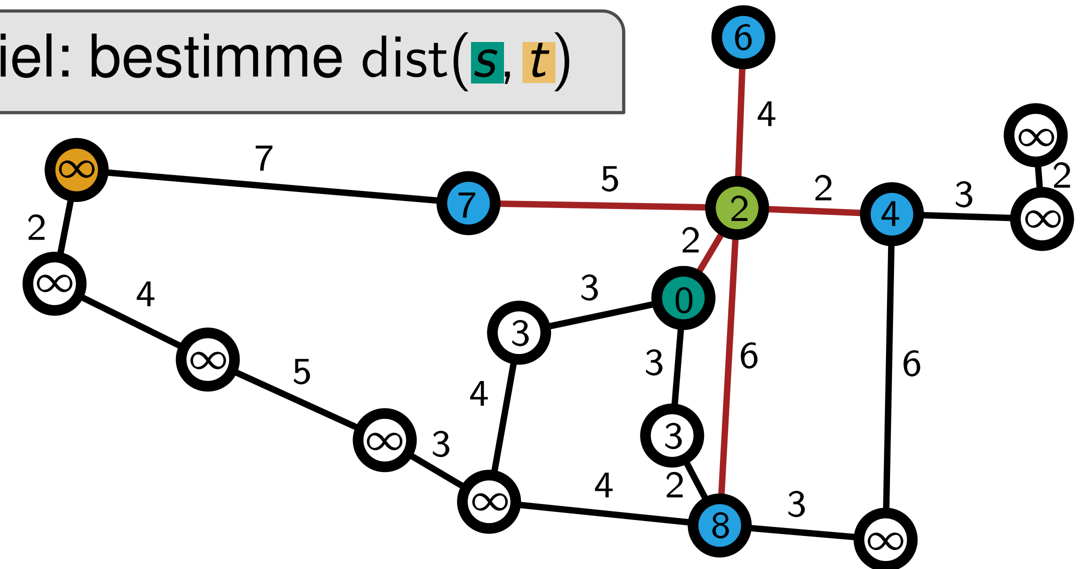
  $u := Q$.**popMin**()

  for *Node* v in $N(u)$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

      Q.**decPrio**(v, $d[v]$)



Dijkstra (Wiederholung)

Dijkstra(*Graph G*, *Node s*)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

PriorityQueue $Q :=$ empty priority queue

for *Node* v in V **do**

  $Q.$**push**$(v, d[v])$

while $Q \neq \emptyset$ **do**

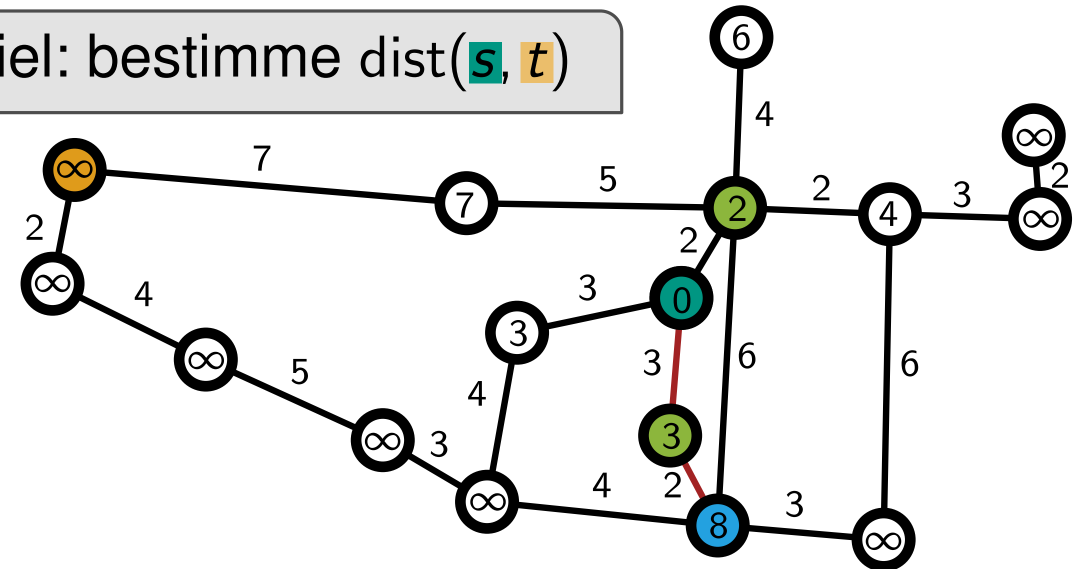
  $u := Q.$**popMin**$()$

  for *Node* v in $N(u)$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

      $Q.$**decPrio**$(v, d[v])$

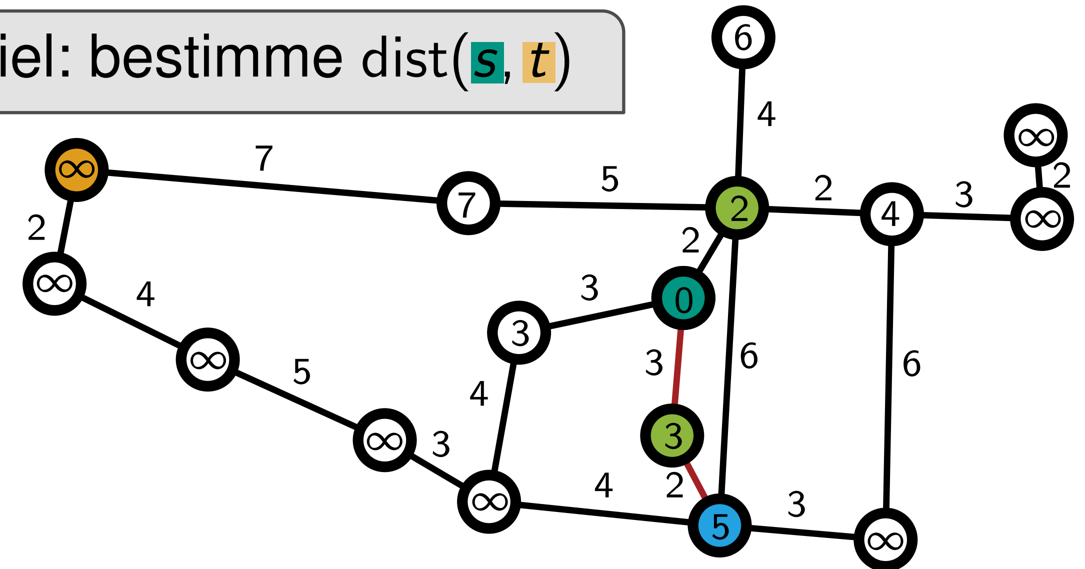


Dijkstra (Wiederholung)

KIT – Karlsruher Institut für Technologie

Dijkstra(*Graph* G, *Node* s)

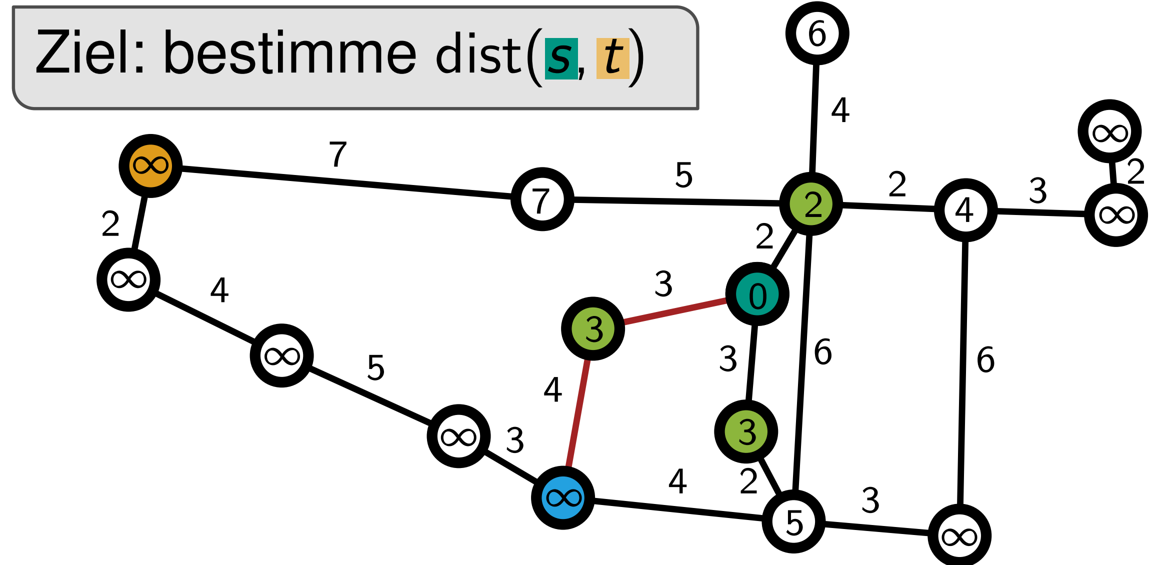
- $d :=$ *Array* of size n initialized with ∞
- $d[s] := 0$
- *PriorityQueue* $Q :=$ empty priority queue
- **for** *Node* v in V **do**
  - Q.**push**$(v, d[v])$
- **while** $Q \neq \emptyset$ **do**
  - $u := Q$.**popMin**()
  - **for** *Node* v in $N(u)$ **do**
    - **if** $d[v] > d[u] + \text{len}(u, v)$ **then**
      - $d[v] := d[u] + \text{len}(u, v)$
      - Q.**decPrio**$(v, d[v])$



Dijkstra (Wiederholung)

Dijkstra(*Graph G*, *Node s*)

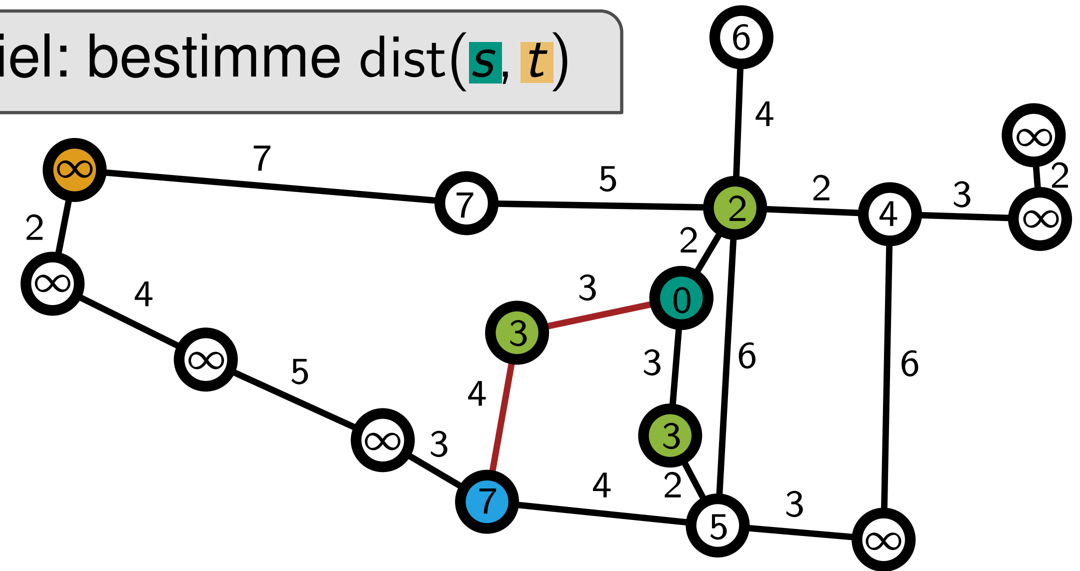
$d :=$ *Array* of size n initialized with ∞
$d[s] := 0$
PriorityQueue $Q :=$ empty priority queue
for *Node* v in V **do**
  Q.**push**$(v, d[v])$
while $Q \neq \emptyset$ **do**
  $u := Q$.**popMin**()
  for *Node* v in $N(u)$ **do**
    if $d[v] > d[u] + \text{len}(u, v)$ **then**
      $d[v] := d[u] + \text{len}(u, v)$
      Q.**decPrio**$(v, d[v])$



Dijkstra (Wiederholung)

Dijkstra(*Graph* G, *Node* s)

- $d :=$ *Array* of size n initialized with ∞
- $d[s] := 0$
- *PriorityQueue* $Q :=$ empty priority queue
- **for** *Node* v in V **do**
  - Q.**push**$(v, d[v])$
- **while** $Q \neq \emptyset$ **do**
  - $u := Q$.**popMin**()
  - **for** *Node* v in $N(u)$ **do**
    - **if** $d[v] > d[u] + \text{len}(u, v)$ **then**
      - $d[v] := d[u] + \text{len}(u, v)$
      - Q.**decPrio**$(v, d[v])$

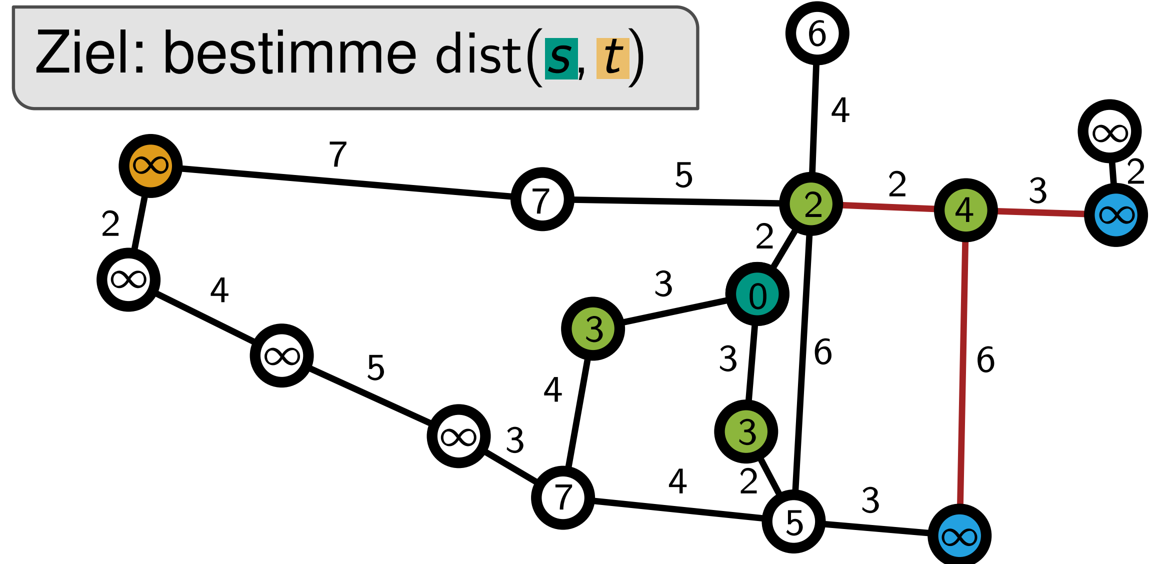


Dijkstra (Wiederholung)

Karlsruher Institut für Technologie

Dijkstra(*Graph* G, *Node* s)

- d := *Array* of size n initialized with ∞
- $d[s] := 0$
- *PriorityQueue* Q := empty priority queue
- **for** *Node* v in V **do**
  - Q.**push**$(v, d[v])$
- **while** $Q \neq \emptyset$ **do**
  - $u := Q$.**popMin**()
  - **for** *Node* v in $N(u)$ **do**
    - **if** $d[v] > d[u] + \text{len}(u, v)$ **then**
      - $d[v] := d[u] + \text{len}(u, v)$
      - Q.**decPrio**$(v, d[v])$



Dijkstra (Wiederholung)

Dijkstra(*Graph G*, *Node s*)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

PriorityQueue Q $:=$ empty priority queue

for *Node* v in V **do**

  Q.**push**$(v, d[v])$

while $Q \neq \emptyset$ **do**

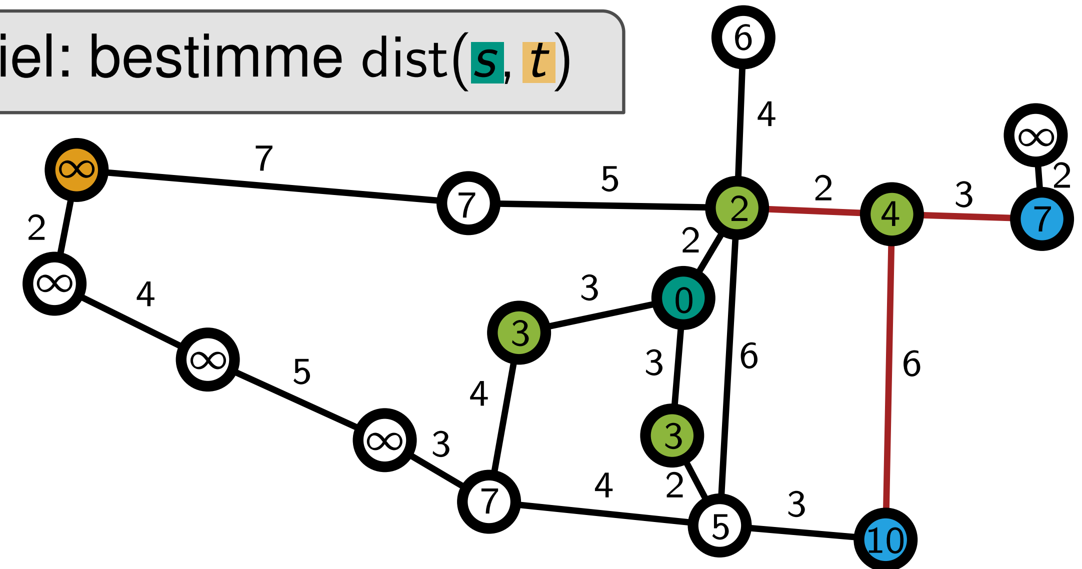
  $u := Q$.**popMin**()

  for *Node* v in $N(u)$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

      Q.**decPrio**$(v, d[v])$



Dijkstra (Wiederholung)

Dijkstra(*Graph G*, *Node s*)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

PriorityQueue $Q :=$ empty priority queue

for *Node* v in V **do**

  Q.**push**$(v, d[v])$

while $Q \neq \emptyset$ **do**

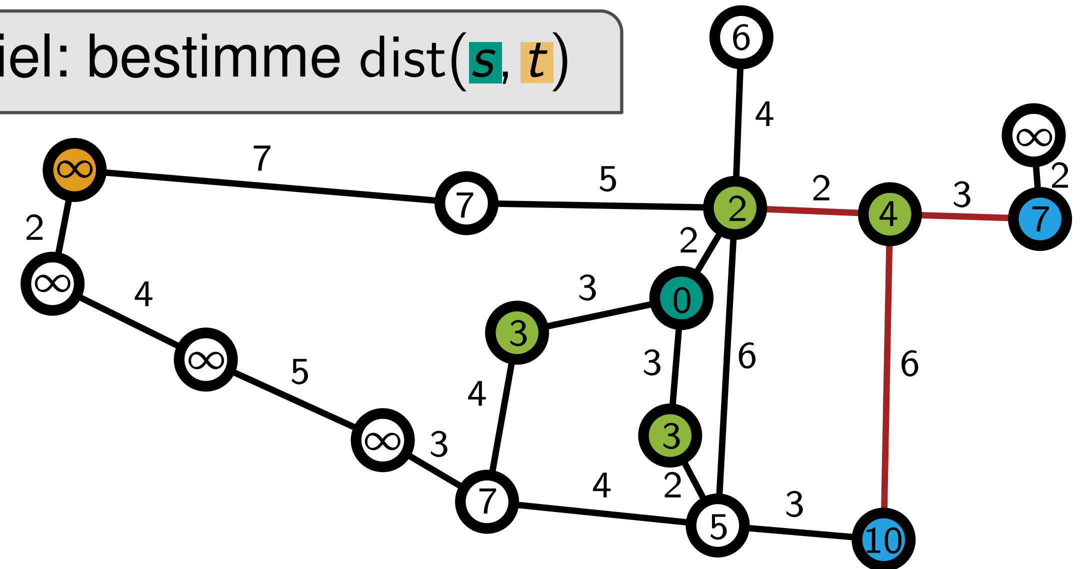
  $u := Q$.**popMin**()

  for *Node* v in

    if $d[v] > d[$

      $d[v] := d$

      Q.**decPr**



Wann ist $\text{dist}(s, t)$ bekannt?

Lemma
In dem Moment, in dem u aus der Queue entfernt und exploriert wird gilt $d[u] = \text{dist}(s, u)$.

Dijkstra (Wiederholung)

Dijkstra(*Graph* G, *Node* s)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

PriorityQueue $Q :=$ empty priority queue

for *Node* v in V **do**

  Q.**push**$(v, d[v])$

while $Q \neq \emptyset$ **do**

  $u := Q$.**popMin**$()$

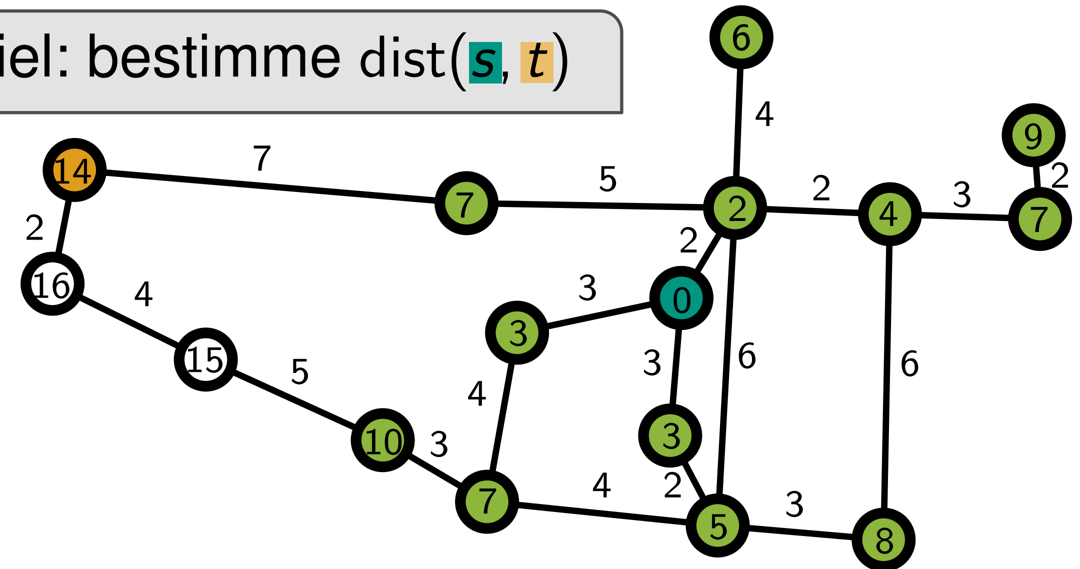
  for *Node* v in $N(u)$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

      Q.**decPrio**$(v, d[v])$

Ziel: bestimme $\text{dist}(s, t)$



Wie viele Knoten werden aus der Queue entfernt, bevor t entfernt wird?

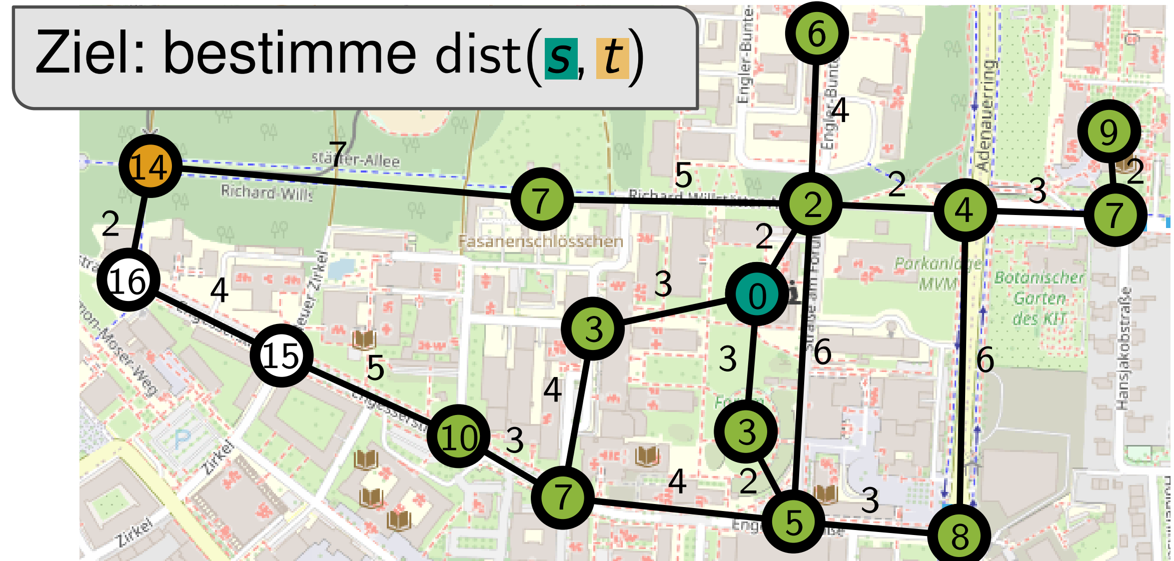
- Knoten werden mit nicht-absteigender Distanz zu s aus der Queue entfernt
- alle Knoten v mit $\text{dist}(s, v) < \text{dist}(s, t)$ werden vor t entfernt

Dijkstra (Wiederholung)

Karlsruher Institut für Technologie

Dijkstra(*Graph* G, *Node* s)

- $d :=$ *Array* of size n initialized with ∞
- $d[s] := 0$
- *PriorityQueue* $Q :=$ empty priority queue
- **for** *Node* v in V **do**
  - Q.**push**$(v, d[v])$
- **while** $Q \neq \emptyset$ **do**
  - $u := Q$.**popMin**()
  - **for** *Node* v in $N(u)$ **do**
    - **if** $d[v] > d[u] + \text{len}(u, v)$ **then**
      - $d[v] := d[u] + \text{len}(u, v)$
      - Q.**decPrio**$(v, d[v])$

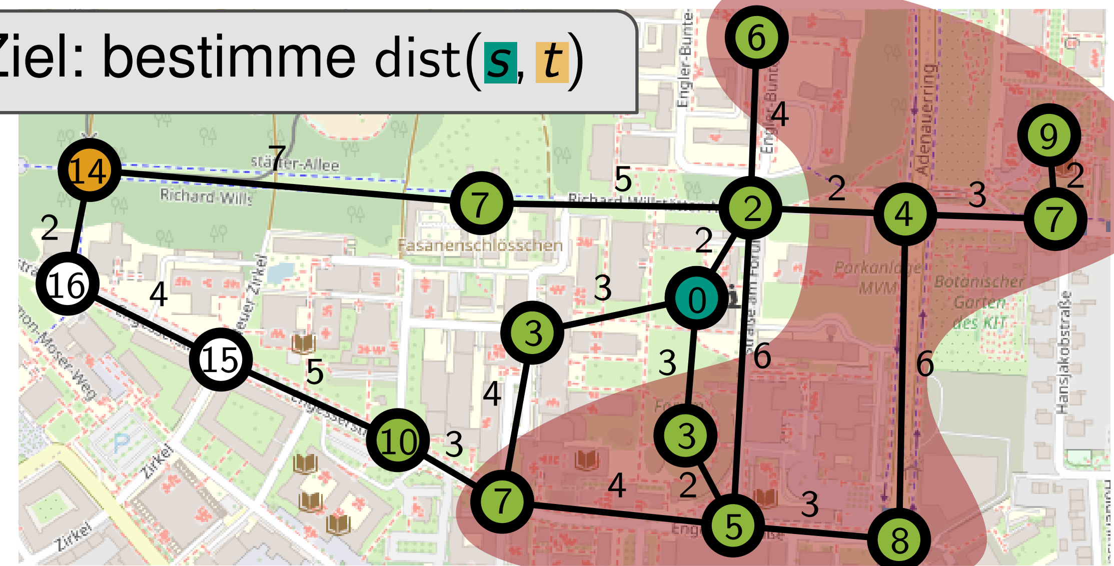


Neues Setting

- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Dijkstra (Wiederholung)

Dijkstra(*Graph* G, *Node* s)
- $d :=$ *Array* of size n initialized with ∞
- $d[s] := 0$
- *PriorityQueue* $Q :=$ empty priority queue
- **for** *Node* v in V **do**
  - Q.**push**$(v, d[v])$
- **while** $Q \neq \emptyset$ **do**
  - $u := Q$.**popMin**()
  - **for** *Node* v in $N(u)$ **do**
    - **if** $d[v] > d[u] + \text{len}(u, v)$ **then**
      - $d[v] := d[u] + \text{len}(u, v)$
      - Q.**decPrio**$(v, d[v])$



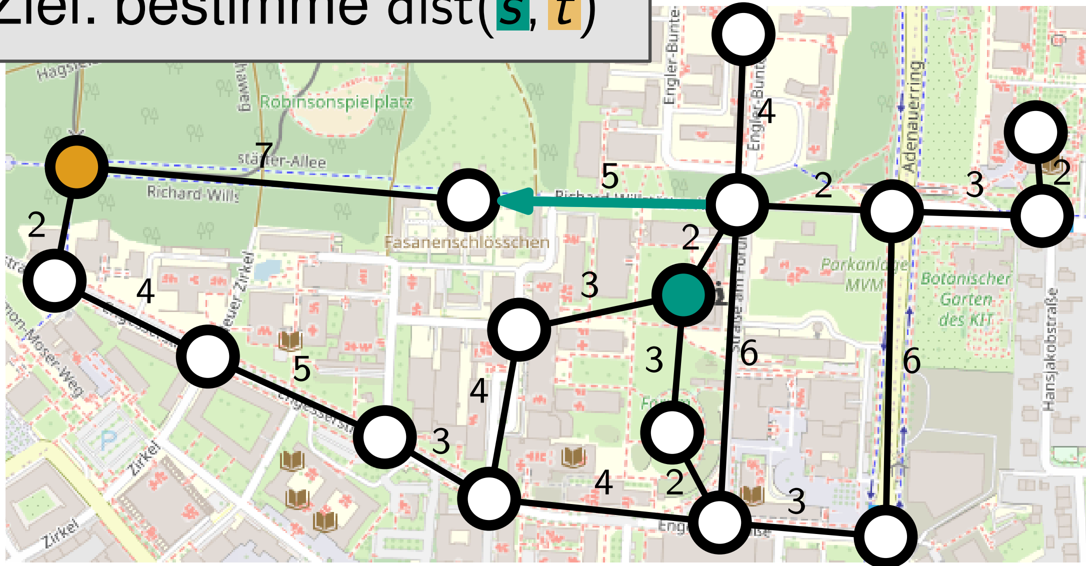
Neues Setting

- Straßengraph mit Straßenlänge als Gewichte (Gewicht 1≙50m)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)
- falsche Richtung?!?

Dijkstra (zielgerichtet)

Karlsruher Institut für Technologie

Ziel: bestimme $\mathrm{dist}(s, t)$



Neues Setting

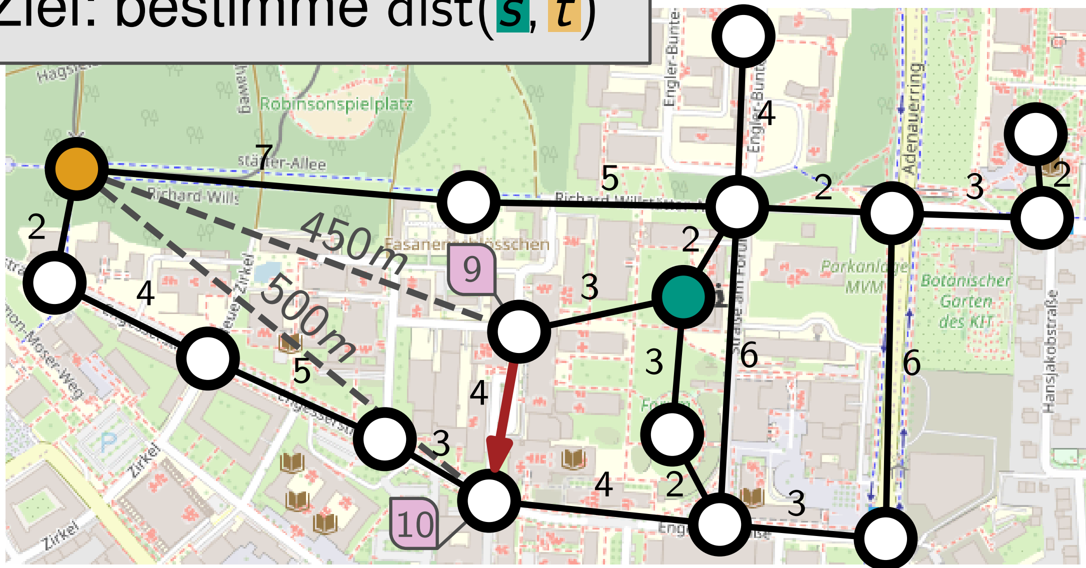
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



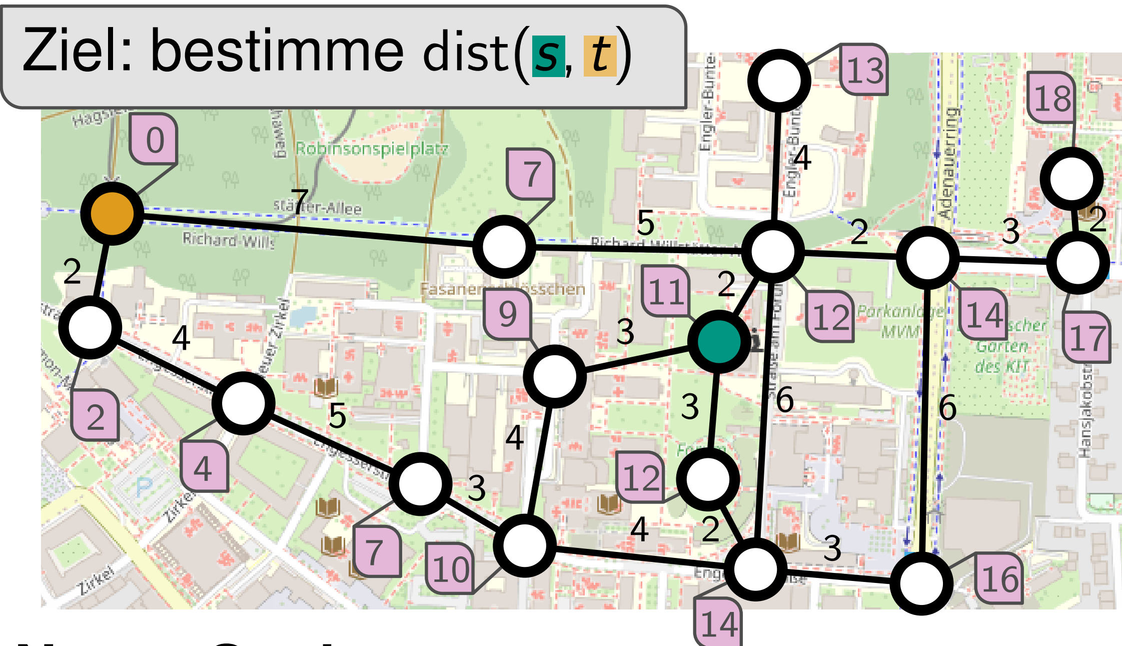
Neues Setting

- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Dijkstra (zielgerichtet)



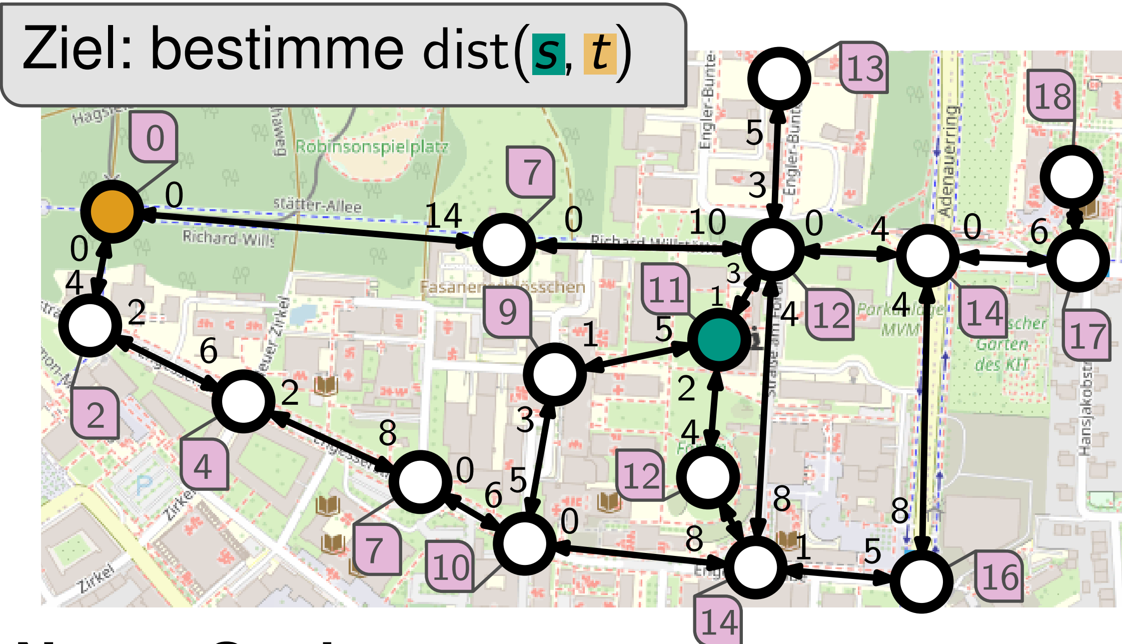
Neues Setting

- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Dijkstra (zielgerichtet)



Neues Setting

- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

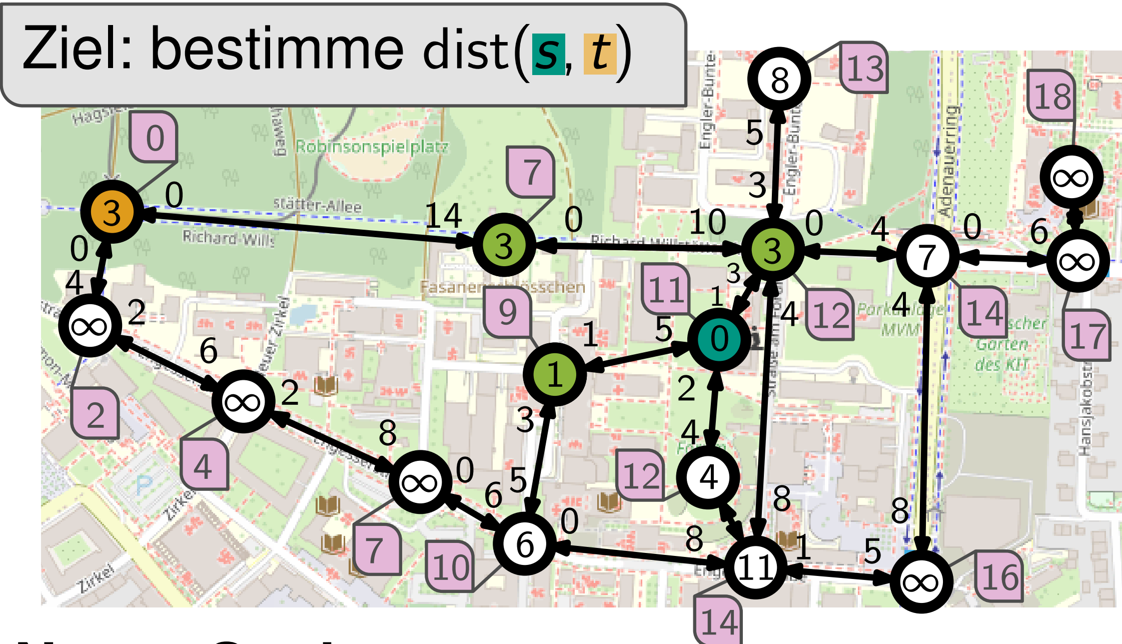
Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

$$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$$

Dijkstra (zielgerichtet)



Neues Setting

- Straßengraph mit Straßenlänge als Gewichte (Gewicht 1≙50m)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

$$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$$

Wie viele Knoten werden aus der Queue entfernt, bevor t entfernt wird?

Bleibt der kürzeste s-t-Pfad gleich?

Dijkstra (zielgerichtet): Korrektheit

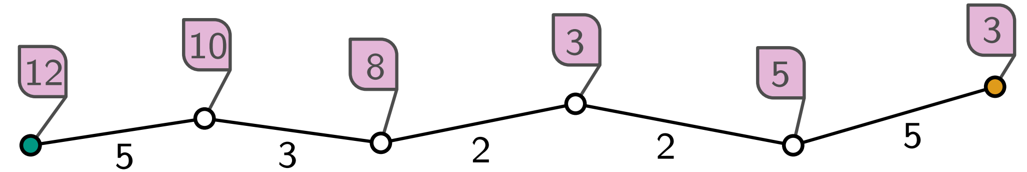
Karlsruher Institut für Technologie

- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

$$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$$

$$\begin{aligned}\text{len}^*(P) &= \sum_{i=1}^{k-1} \text{len}^*(v_i, v_{i+1}) \\ &= \sum_{i=1}^{k-1} (\text{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\ &= \sum_{i=1}^{k-1} \text{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1}))\end{aligned}$$



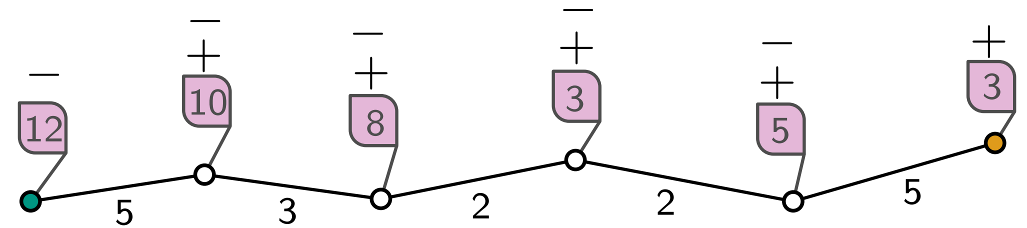
Dijkstra (zielgerichtet): Korrektheit

Karlsruher Institut für Technologie

- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

$\mathrm{len}^*(u, v) = \mathrm{len}(u, v) - \pi(u) + \pi(v)$



$$\begin{aligned}\mathrm{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \mathrm{len}^*(v_i, v_{i+1}) \\ &= \textstyle\sum_{i=1}^{k-1} (\mathrm{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\ &= \textstyle\sum_{i=1}^{k-1} \mathrm{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1}))\end{aligned}$$

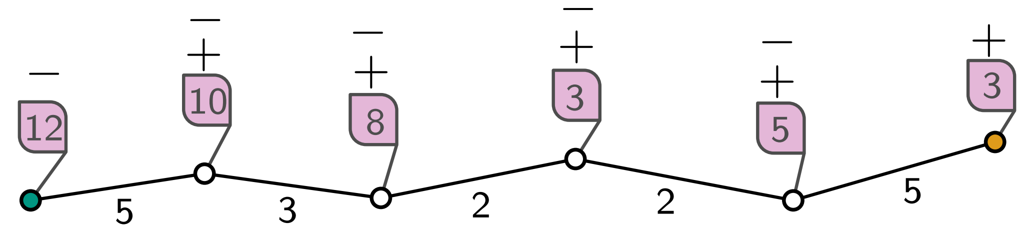
Dijkstra (zielgerichtet): Korrektheit

Karlsruher Institut für Technologie

- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

$$\text{len}^*(u,v) = \text{len}(u,v) - \pi(u) + \pi(v)$$



$$\begin{aligned}
\text{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \text{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\text{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \text{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1})) \\
&= \text{len}(P) - \pi(v_1) + \pi(v_2) - \pi(v_2) + \pi(v_3) - \pi(v_3) + \pi(v_4) - \ldots - \pi(v_{k-1}) + \pi(v_k)
\end{aligned}$$

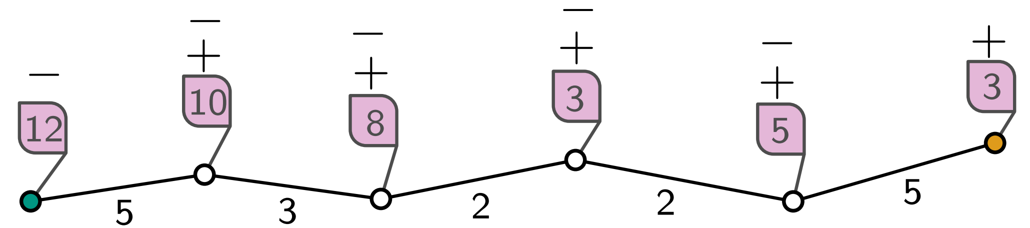
Dijkstra (zielgerichtet): Korrektheit

Karlsruher Institut für Technologie

- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

$$\text{len}^*(u,v) = \text{len}(u,v) - \pi(u) + \pi(v)$$



$$
\begin{aligned}
\text{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \text{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\text{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \text{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1})) \\
&= \text{len}(P) - \pi(v_1) + \cancel{\pi(v_2)} - \cancel{\pi(v_2)} + \cancel{\pi(v_3)} - \cancel{\pi(v_3)} + \cancel{\pi(v_4)} - \ldots - \cancel{\pi(v_{k-1})} + \pi(v_k) \\
&= \text{len}(P) - \pi(v_1) + \pi(v_k) \\
&= \text{len}(P) - \pi(s) + \pi(t)
\end{aligned}
$$

Dijkstra (zielgerichtet): Korrektheit

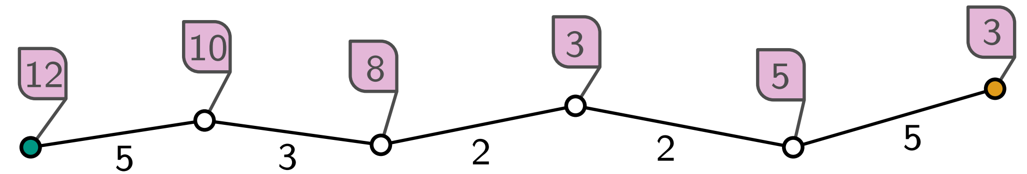
Karlsruher Institut für Technologie

- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \dots, v_k = t)$

$$\text{len}^*(P) = \text{len}(P) \underbrace{- \pi(s) + \pi(t)}_{\text{konstant für alle } s\text{-}t\text{-Pfade}}$$

Neue Kantengewichtsfunktion in G^*

$$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$$



Dijkstra (zielgerichtet): Korrektheit

- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

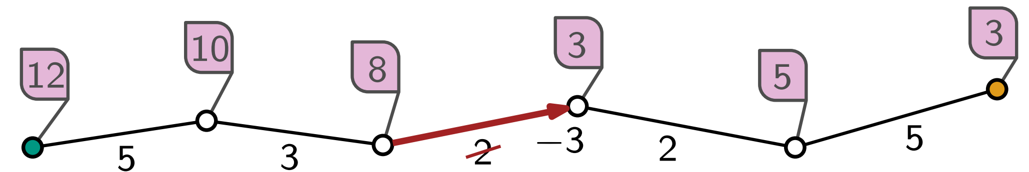
$$\text{len}^*(P) = \text{len}(P) \underbrace{- \pi(s) + \pi(t)}_{\text{konstant für alle } s\text{-}t\text{-Pfade}}$$

- Formel gilt für alle Potentialfunktionen $\pi : V \to \mathbb{R}$

Neue Kantengewichtsfunktion in G^*

$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$

↯ negative Kantengewichte



Findet Dijkstra mit jedem π den kürzesten s-t-Weg?

Dijkstra (zielgerichtet): Korrektheit

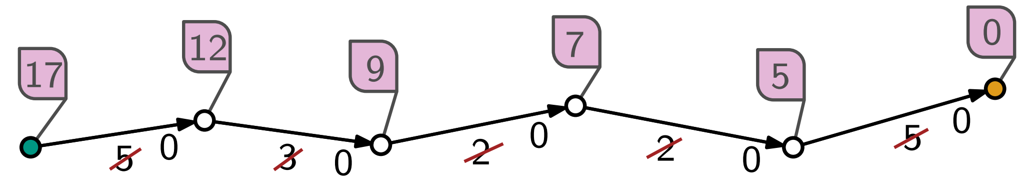
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

$$\text{len}^*(P) = \text{len}(P) \underbrace{- \pi(s) + \pi(t)}_{\text{konstant für alle } s\text{-}t\text{-Pfade}}$$

- Formel gilt für alle Potentialfunktionen $\pi : V \to \mathbb{R}$
- gute Potentialfunktion:
  - $\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v) \geq 0$ für alle $(u, v) \in E$
  - $\pi(u)$ möglichst nah an $\text{dist}(u, t)$

Neue Kantengewichtsfunktion in G^*

$$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$$



Findet Dijkstra mit jedem π den kürzesten s-t-Weg?

Ist euklidische Distanz auf Straßengraphen eine gute Potentialfunktion?

Was passiert bei $\pi(u) = \text{dist}(u, t)$ für alle $u \in V$?

Dijkstra (zielgerichtet): Korrektheit

Karlsruher Institut für Technologie

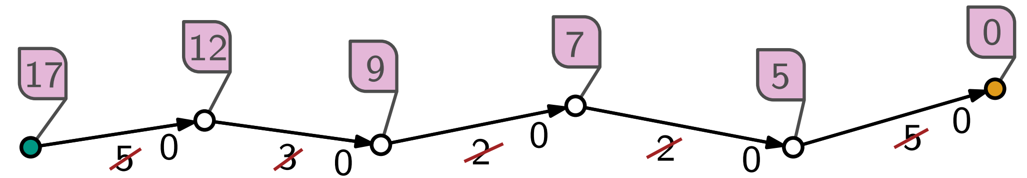
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \dots, v_k = t)$

$$\text{len}^*(P) = \text{len}(P) \underbrace{- \pi(s) + \pi(t)}_{\text{konstant für alle } s\text{-}t\text{-Pfade}}$$

- Formel gilt für alle Potentialfunktionen $\pi : V \to \mathbb{R}$
- gute Potentialfunktion:
  - $\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v) \geq 0$ für alle $(u, v) \in E$
  - $\pi(u)$ möglichst nah an $\text{dist}(u, t)$
- oft bessere Laufzeit in der Praxis (NICHT im Worst Case)

Neue Kantengewichtsfunktion in G^*

$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$



Findet Dijkstra mit jedem π den kürzesten s-t-Weg?

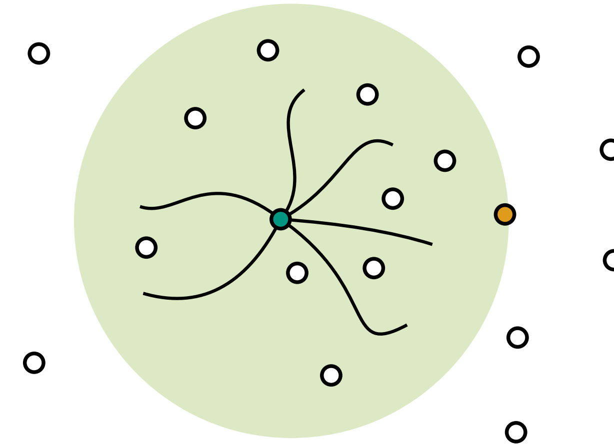
Ist euklidische Distanz auf Straßengraphen eine gute Potentialfunktion?

Was passiert bei $\pi(u) = \text{dist}(u, t)$ für alle $u \in V$?

Dijkstra (zielgerichtet)

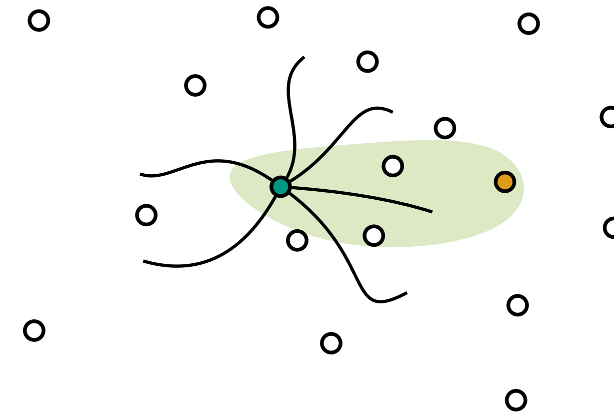
Karlsruher Institut für Technologie

- Dijkstra sucht viel ab, um Ziel zu finden



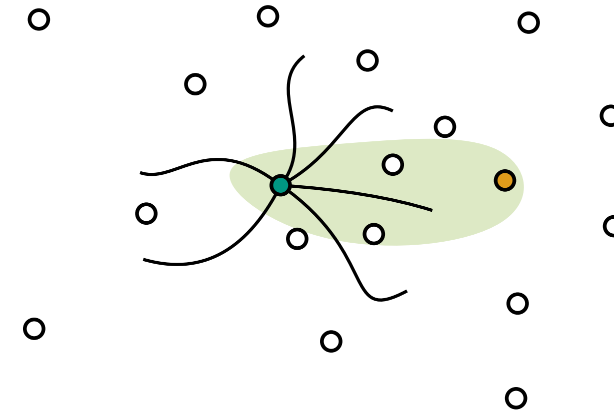
Dijkstra (zielgerichtet)

- Dijkstra sucht viel ab, um Ziel zu finden
- Potentialfunktion: schätzt, wie gut ein Knoten ist, wenn man zum Ziel möchte
- auf modifiziertem Graph: kleinerer Suchraum



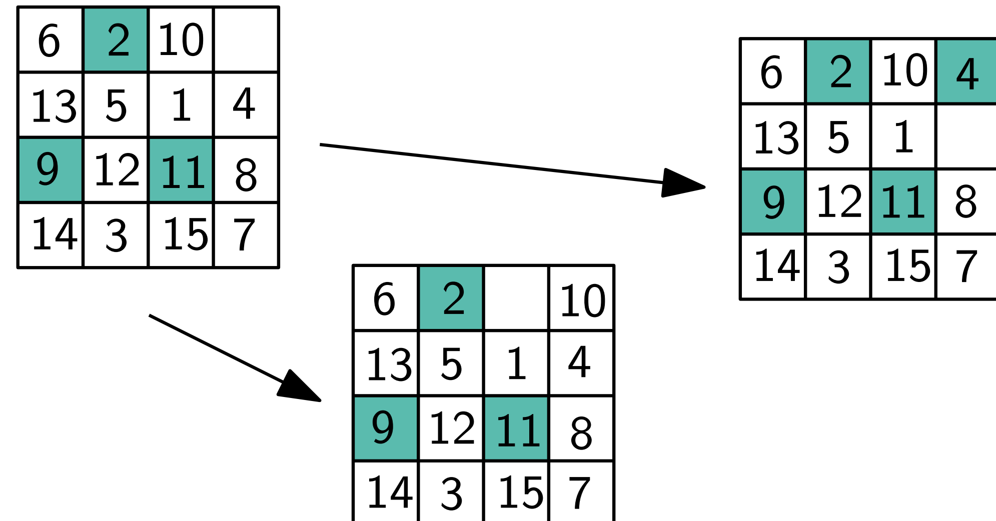
Dijkstra (zielgerichtet)

- Dijkstra sucht viel ab, um Ziel zu finden
- Potentialfunktion: schätzt, wie gut ein Knoten ist, wenn man zum Ziel möchte
- auf modifiziertem Graph: kleinerer Suchraum



Anwendungen

- verallgemeinerte Variante: A*
- Routenplanung
- Robotik
- Puzzle lösen
  - Potential: Anzahl richtige Steine



Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Packag
Descrip
Upstrea
License
Conflict
Submitt
Maintai
Last Pa
Votes:
Popular
First Su
Last Up

Depend

freetyp
freetyp
gsl (*gsl-*
hicolor-
libspiro
lua53
poppler
qt5-bas
qt5-svg
zlib (*zli*
git)

poppler 22.06.0-1

Architecture: x86_64

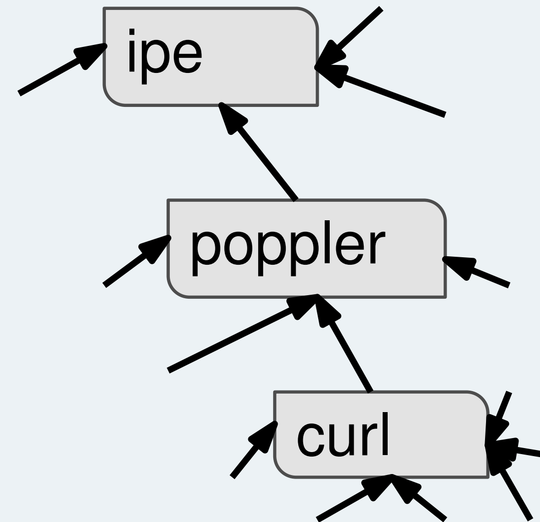
Reposito
Split Pacl
Descripti
Upstream
License(s
Provides:
Conflicts:
Maintain
Package
Installed
Last Pack
Build Dat
Signed B
Signature
Last Upd

Depende

cairo
curl
fontconfi
gcc-libs
lcms2

curl 7.83.1-1

Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian Hesse
Build Date:	2022-05-11 06:34 UTC
Signed By:	Christian Hesse
Signature Date:	2022-05-11 06:41 UTC
Last Updated:	2022-05-11 15:10 UTC



Dependencies (16)

brotli
ca-certificates
krb5
libbrotlidec.so=1-64 (*brotli*)
libgssapi_krb5.so=2-64 (*krb5*)
libidn2

Required By (401)

0ad
appstream
arch-audit
archlinux-repro
ardour (*requires libcurl.so*)
aria

Problem: Dependencies

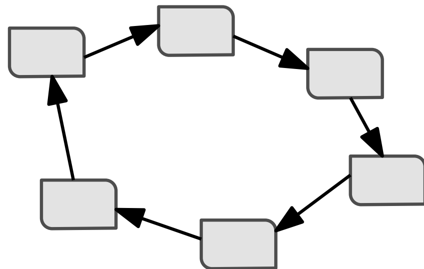
Karlsruher Institut für Technologie

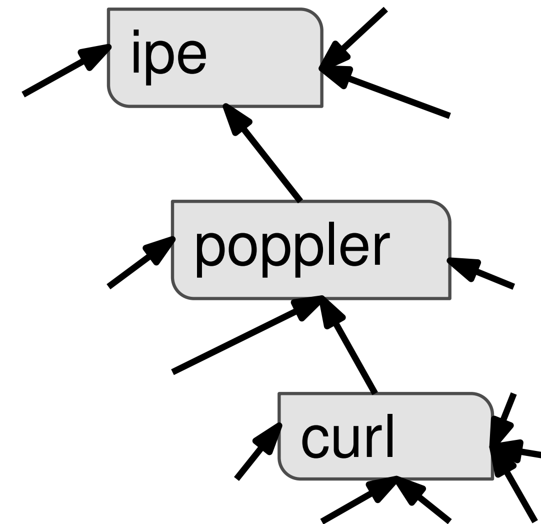
Modellierung als Graph

- Knoten V: Menge von Paketen
- Knoten E: $(v, w) \in E \Leftrightarrow v$ von w benötigt

Frage

Gibt es zyklische Abhängigkeiten?





Cyclic Dependency

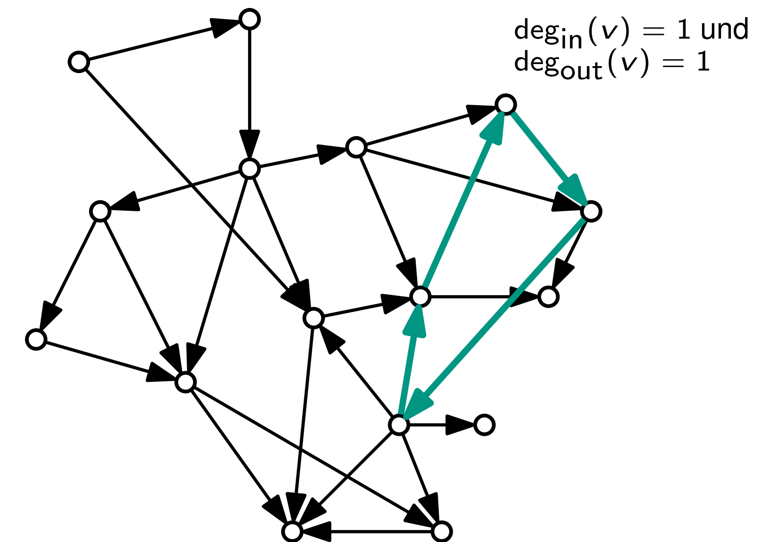
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$ $O(2^m)$
    - prüfe ob E' Kreis ergibt $O(n+m)$

Gesamt: $O(2^m \cdot (n+m))$



Cyclic Dependency

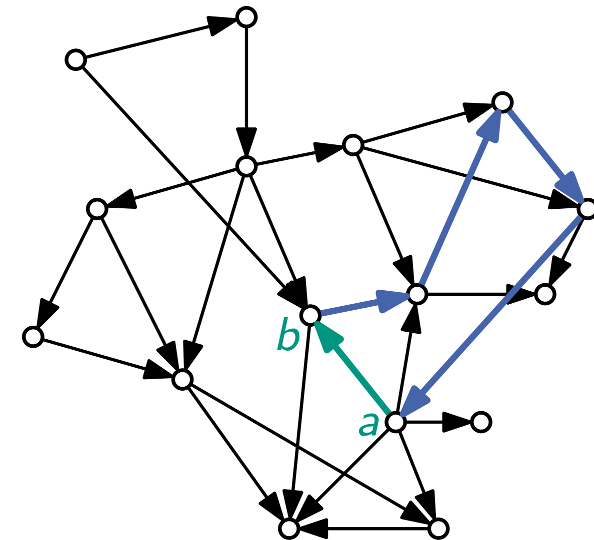
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $O(m)$
    - suche b-a-Pfad $O(n+m)$

Gesamt: $O(m(n+m))$



Frage: Ist Linearzeit möglich?

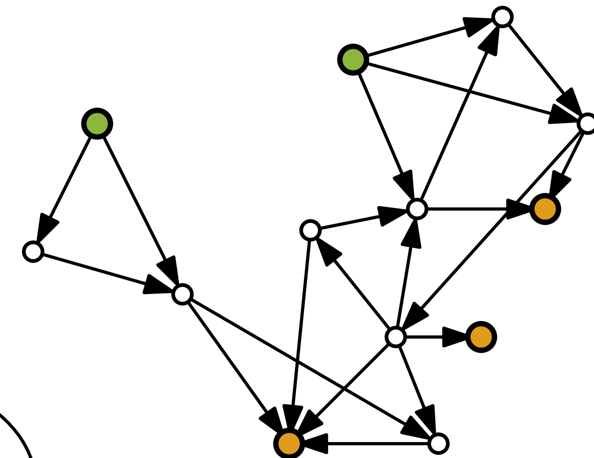
Cyclic Dependency

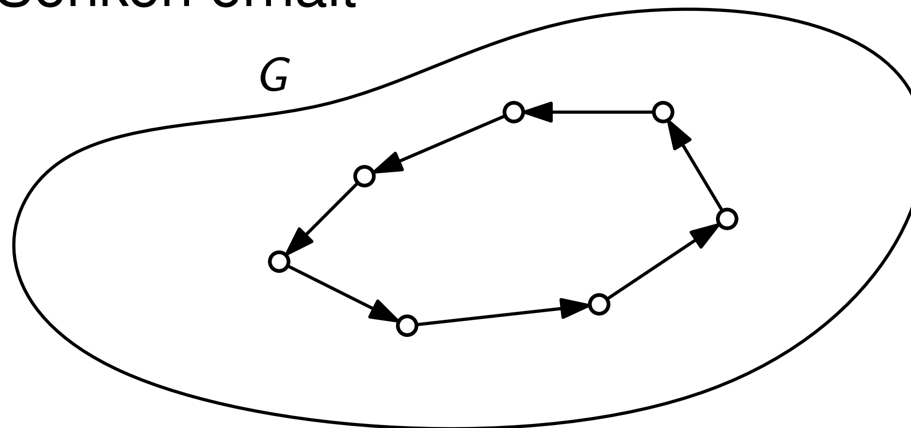
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)





Cyclic Dependency

Problemstellung

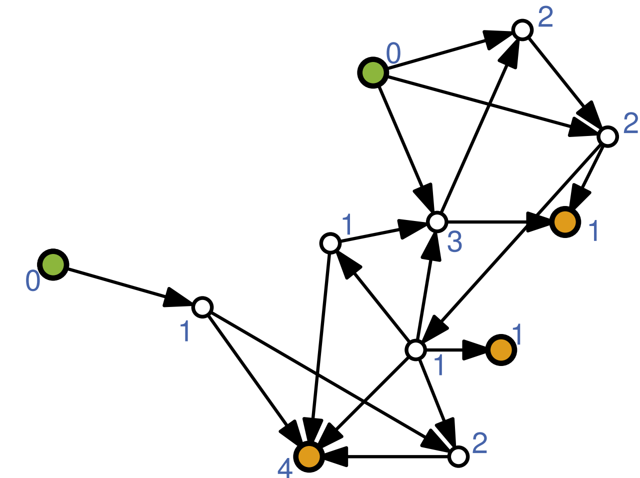
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{out}(v))$?
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{out}(v))$



$\deg_{in} : [\mathbb{N}] = [0, 0, 1, 0, \ldots]$

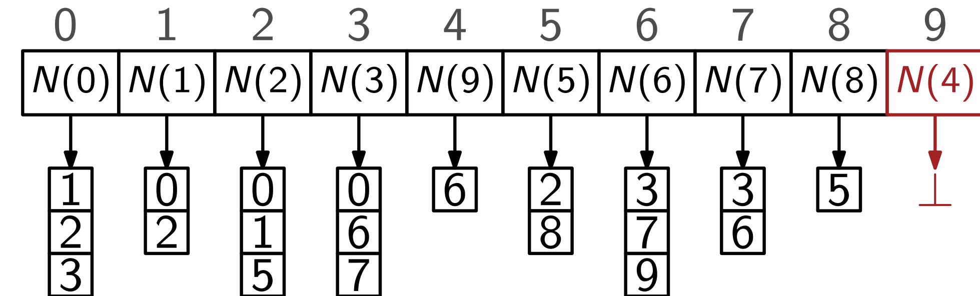
sources : Queue

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

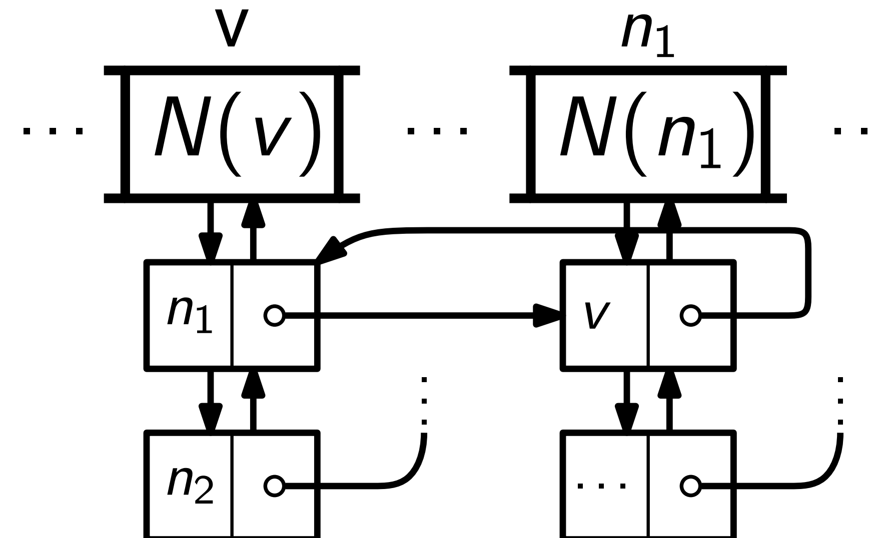
- Knoten
- Kanten zu gelöschten Knoten ✓ teuer
- Kanten von gelöschten Knoten ✓



Beispiel: Lösche 4

Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$



Knoten löschen

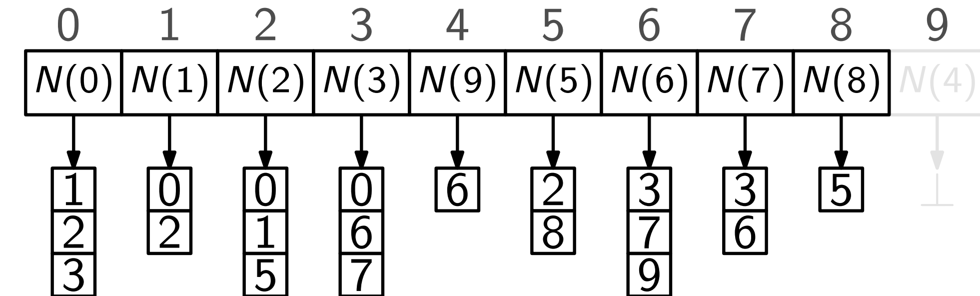
Karlsruher Institut für Technologie

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten ✓
- Kanten von gelöschten Knoten ✓

teuer



Beispiel: Lösche 4

Aktualisierung Indizes:

`new_index`[9] = 4
`old_index`[4] = 9

Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n - 1$
  - mapping zw. neuen und alten Indizes `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$

Gesamtlaufzeit: $\Theta(\deg(v))$

Und auf gerichteten Graphen?

Knoten löschen?

Karlsruher Institut für Technologie

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

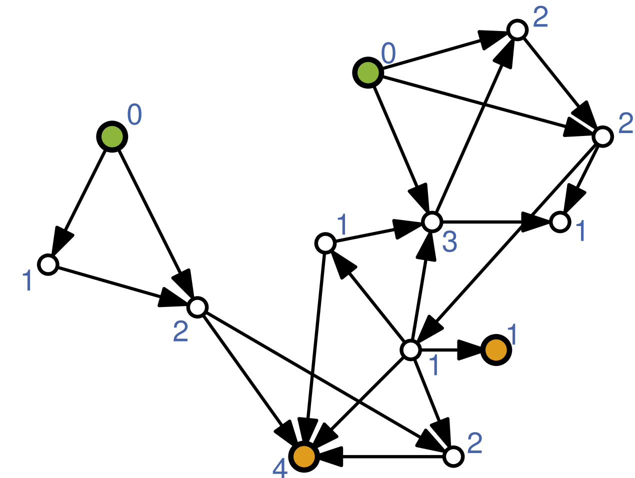
Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$

Wichtig

- Knoten *löschen* oft nicht notwendig
- geschicktes Verwalten zusätzlicher Informationen hilfreich



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue