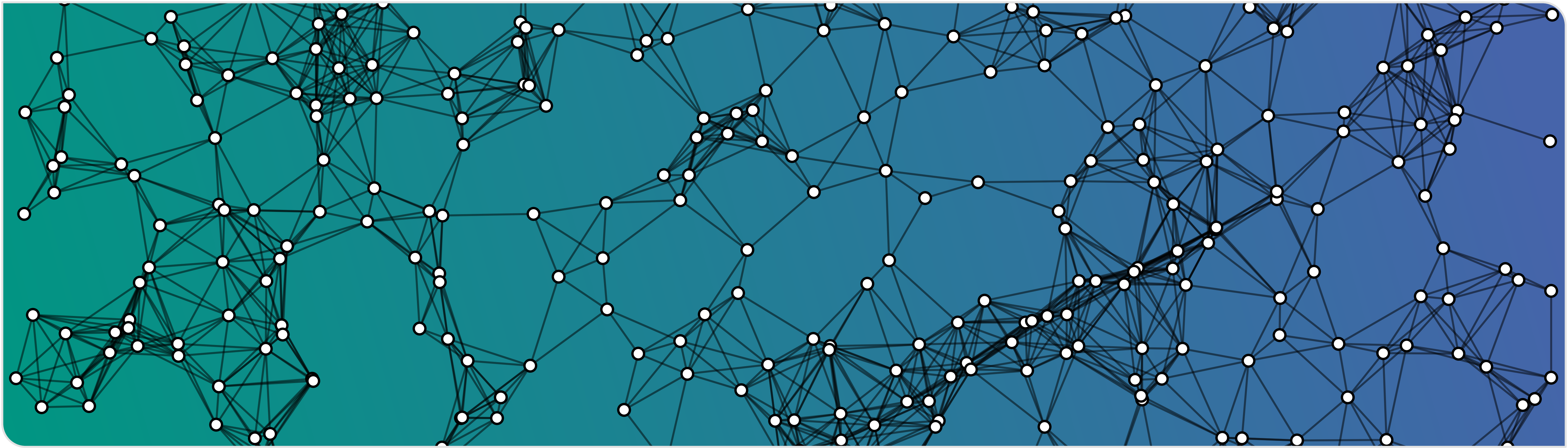
Karlsruher Institut für Technologie

Algorithmen 1

Übung 4
Graphen, kürzeste Wege

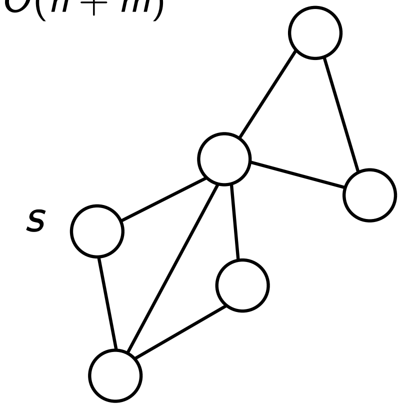


Kürzeste-Wege-Algorithmen

Kürzeste-Wege-Algorithmen

Breitensuche

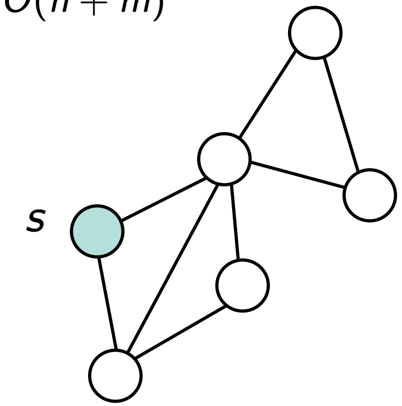
$O(n + m)$



Kürzeste-Wege-Algorithmen

Breitensuche

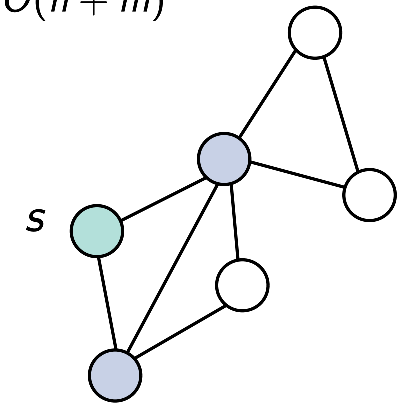
$O(n + m)$



Kürzeste-Wege-Algorithmen

Breitensuche

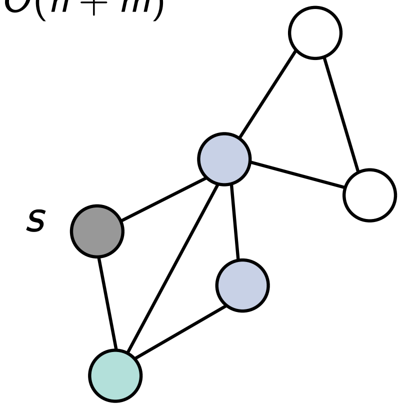
$O(n + m)$



Kürzeste-Wege-Algorithmen

Breitensuche

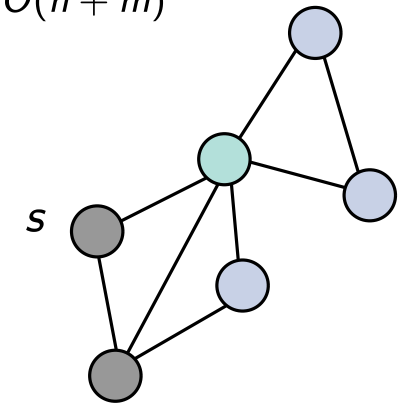
$O(n + m)$



Kürzeste-Wege-Algorithmen

Breitensuche

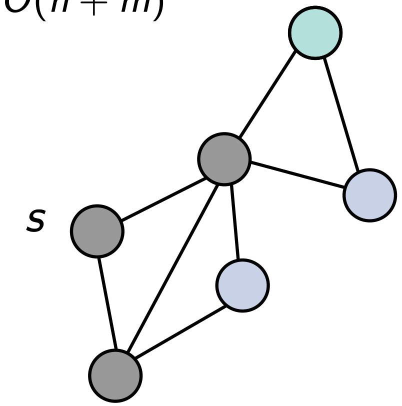
$O(n + m)$



Kürzeste-Wege-Algorithmen

Breitensuche

$O(n + m)$

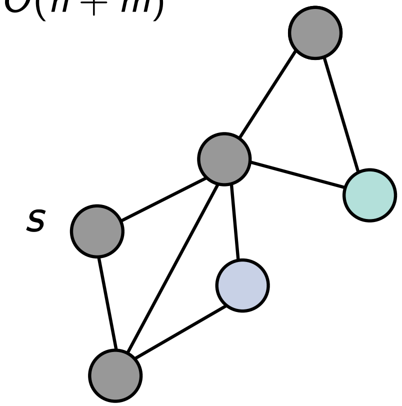


Kürzeste-Wege-Algorithmen

Karlsruher Institut für Technologie

Breitensuche

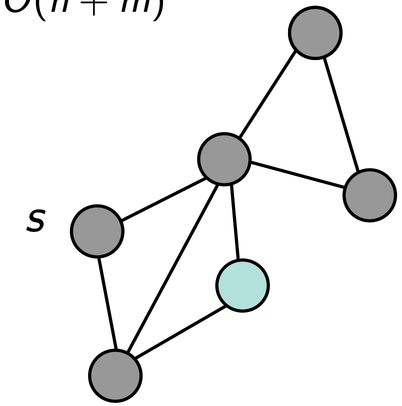
$O(n + m)$



Kürzeste-Wege-Algorithmen

Breitensuche

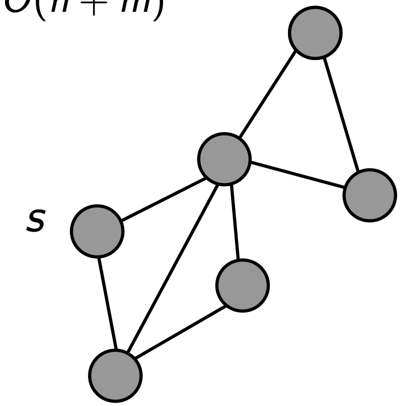
$O(n + m)$



Kürzeste-Wege-Algorithmen

Breitensuche

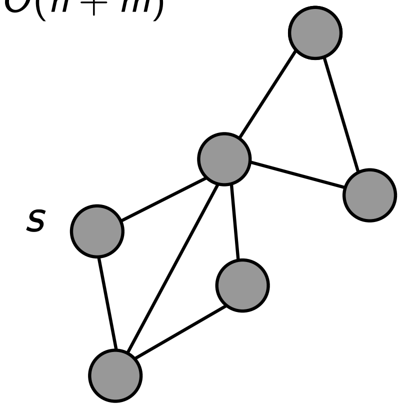
$O(n + m)$



Kürzeste-Wege-Algorithmen

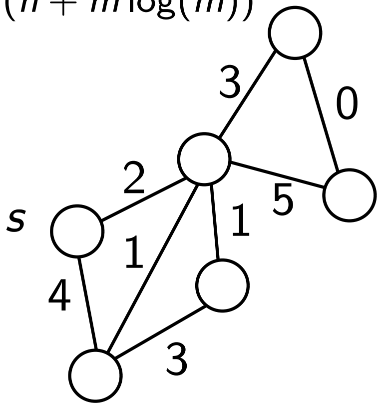
Breitensuche

$O(n + m)$



Dijkstra

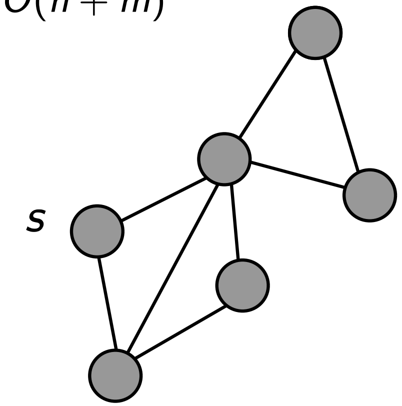
$O(n + m \log(m))$



Kürzeste-Wege-Algorithmen

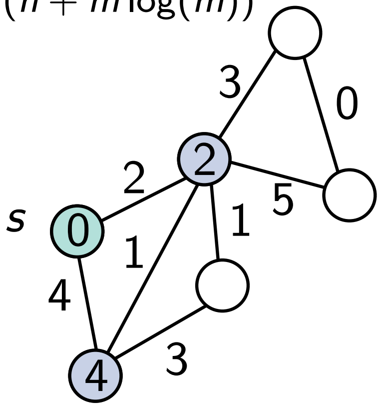
Breitensuche

$O(n + m)$



Dijkstra

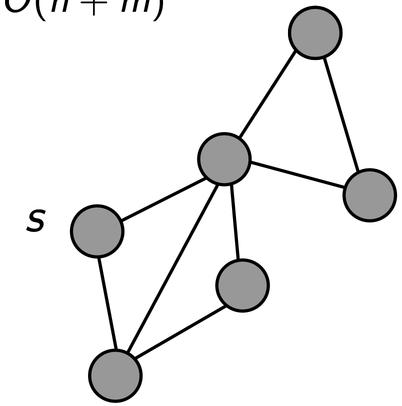
$O(n + m \log(m))$



Kürzeste-Wege-Algorithmen

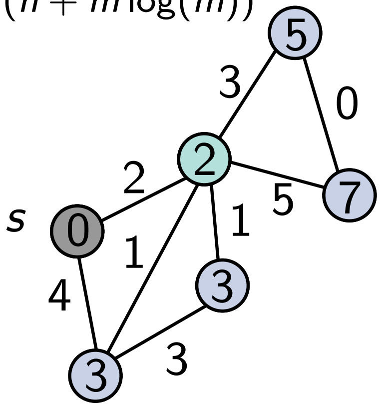
Breitensuche

$O(n+m)$



Dijkstra

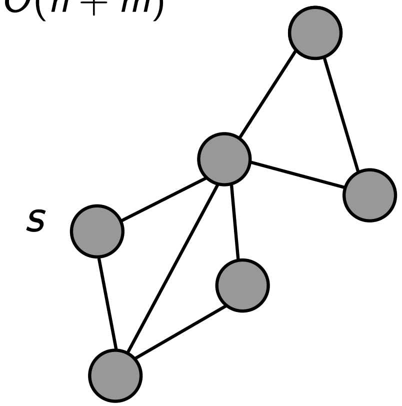
$O(n+m\log(m))$



Kürzeste-Wege-Algorithmen

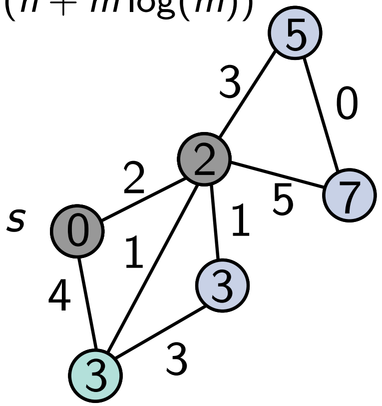
Breitensuche

$O(n + m)$



Dijkstra

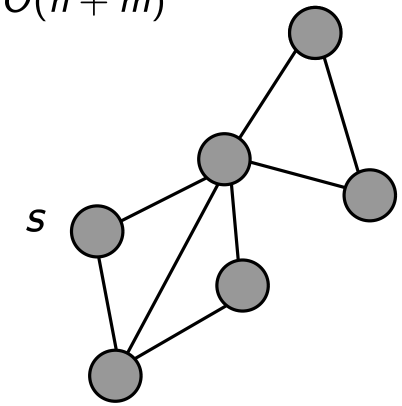
$O(n + m \log(m))$



Kürzeste-Wege-Algorithmen

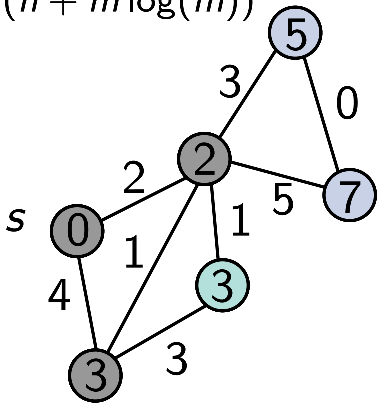
Breitensuche

$O(n + m)$



Dijkstra

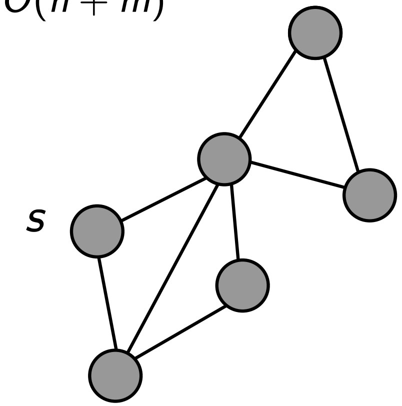
$O(n + m \log(m))$



Kürzeste-Wege-Algorithmen

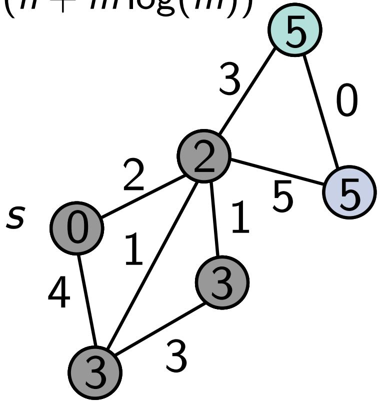
Breitensuche

$O(n+m)$



Dijkstra

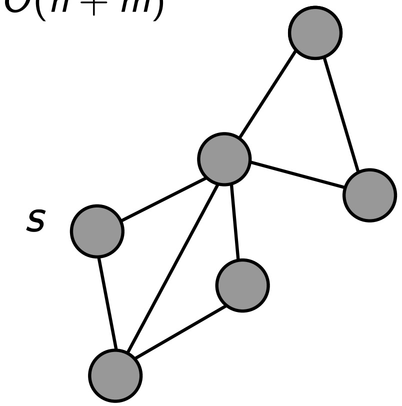
$O(n+m\log(m))$



Kürzeste-Wege-Algorithmen

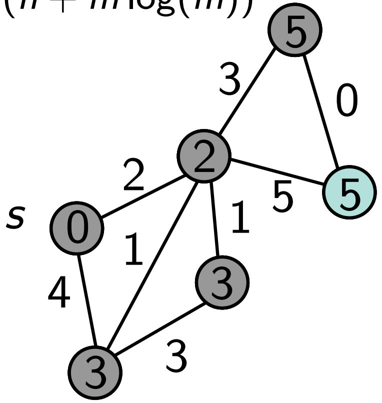
Breitensuche

$O(n + m)$



Dijkstra

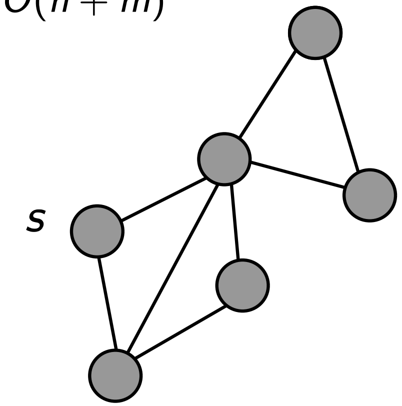
$O(n + m \log(m))$



Kürzeste-Wege-Algorithmen

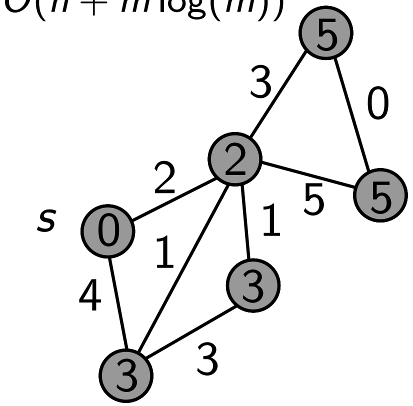
Breitensuche

$O(n+m)$



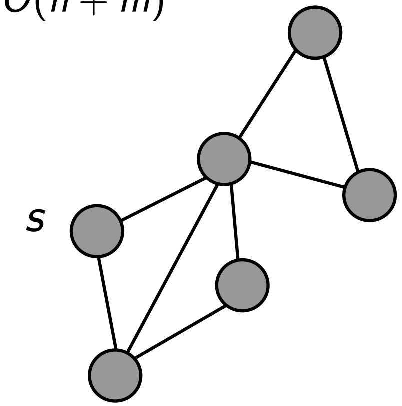
Dijkstra

$O(n+m\log(m))$

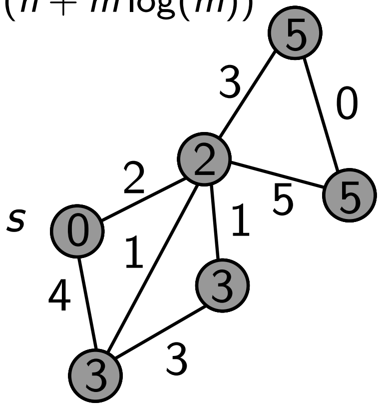


Kürzeste-Wege-Algorithmen

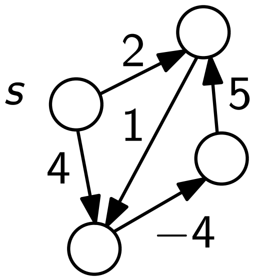
Breitensuche
$O(n + m)$



Dijkstra
$O(n + m\log(m))$

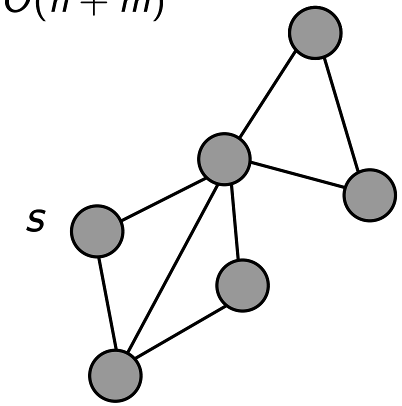


Bellman-Ford
$O(nm)$

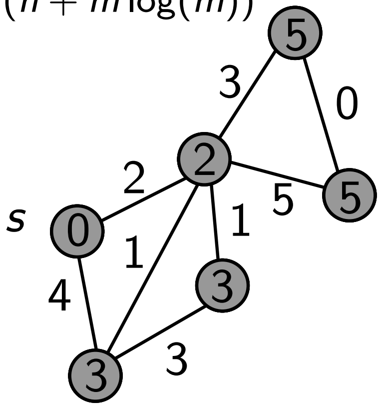


Kürzeste-Wege-Algorithmen

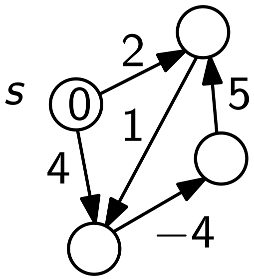
Breitensuche
$O(n + m)$



Dijkstra
$O(n + m \log(m))$

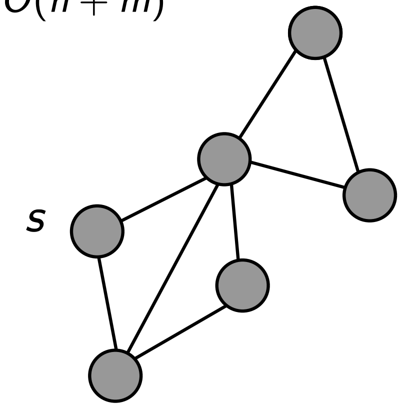


Bellman-Ford
$O(nm)$

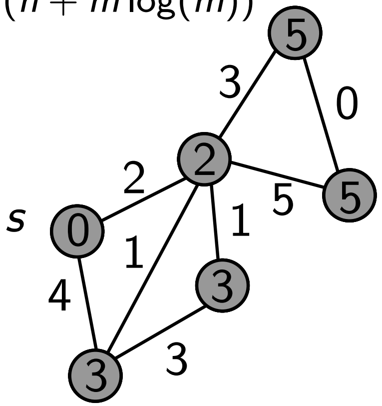


Kürzeste-Wege-Algorithmen

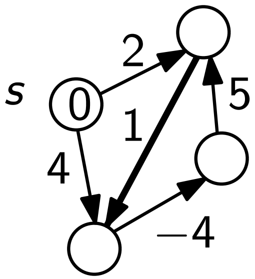
Breitensuche
$O(n+m)$



Dijkstra
$O(n + m\log(m))$

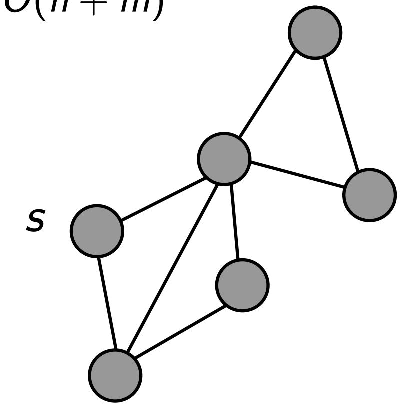


Bellman-Ford
$O(nm)$

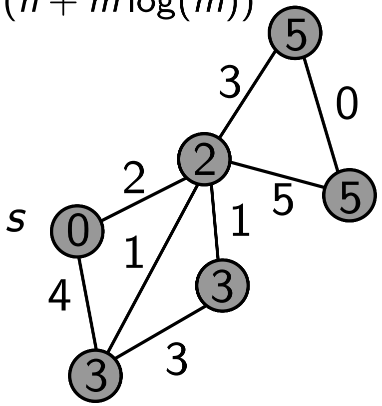


Kürzeste-Wege-Algorithmen

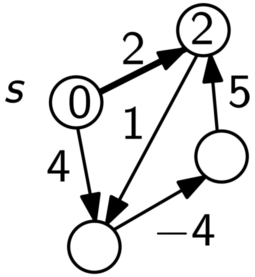
Breitensuche
$O(n+m)$



Dijkstra
$O(n + m\log(m))$



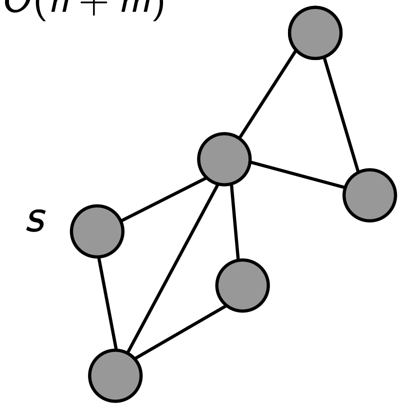
Bellman-Ford
$O(nm)$



Kürzeste-Wege-Algorithmen

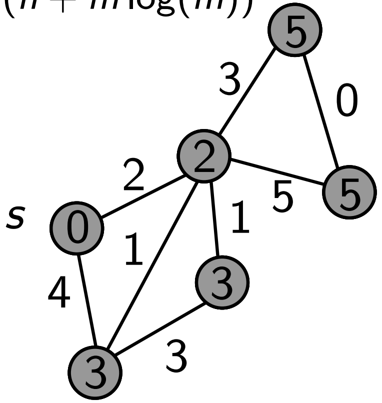
Breitensuche

$O(n + m)$



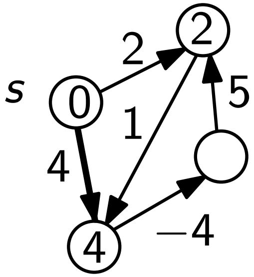
Dijkstra

$O(n + m\log(m))$



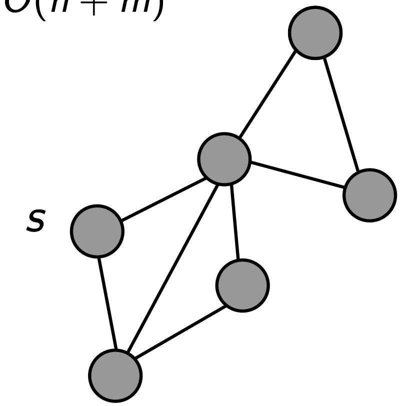
Bellman-Ford

$O(nm)$

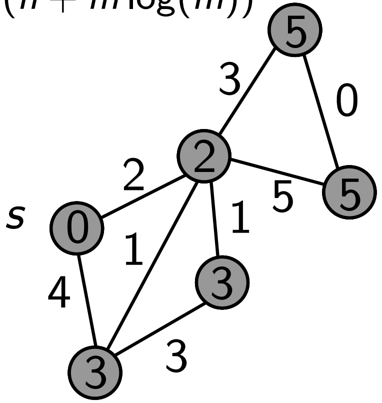


Kürzeste-Wege-Algorithmen

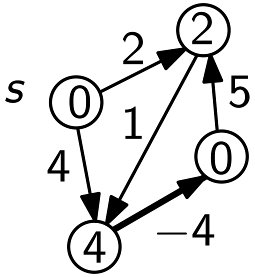
Breitensuche
$O(n+m)$



Dijkstra
$O(n + m\log(m))$

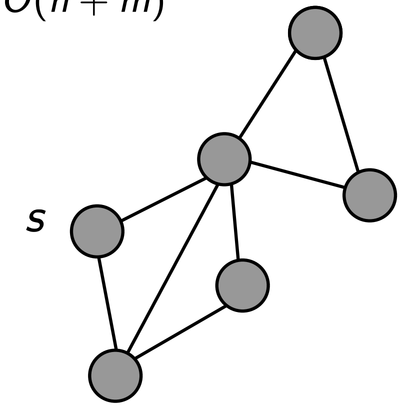


Bellman-Ford
$O(nm)$

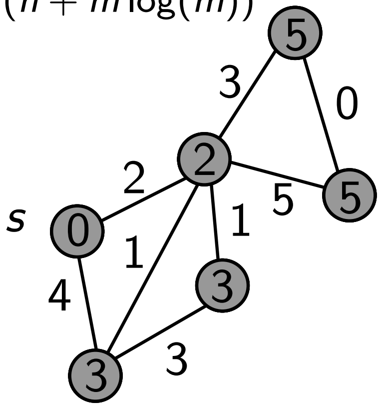


Kürzeste-Wege-Algorithmen

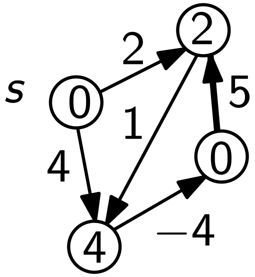
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



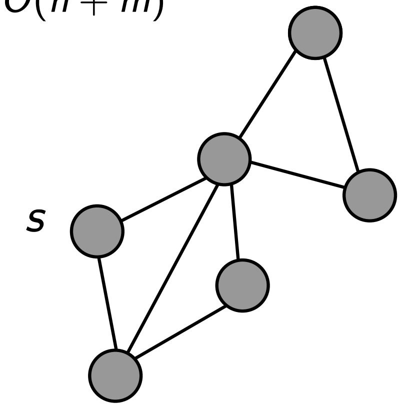
Bellman-Ford
$O(nm)$



Kürzeste-Wege-Algorithmen

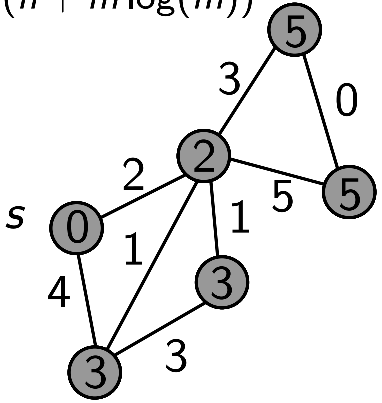
Breitensuche

$O(n+m)$



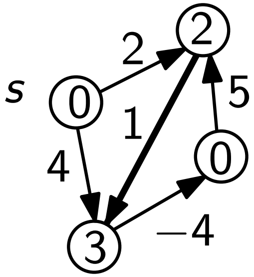
Dijkstra

$O(n+m\log(m))$



Bellman-Ford

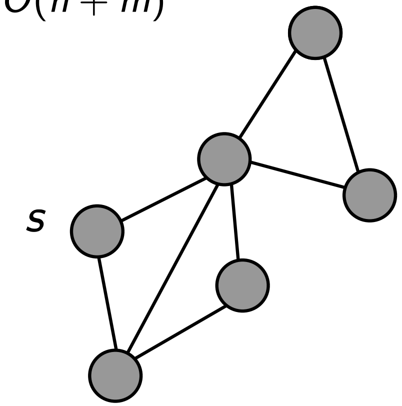
$O(nm)$



Kürzeste-Wege-Algorithmen

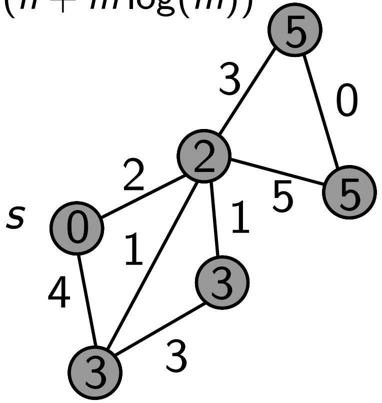
Breitensuche

$O(n + m)$



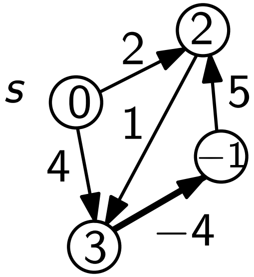
Dijkstra

$O(n + m \log(m))$



Bellman-Ford

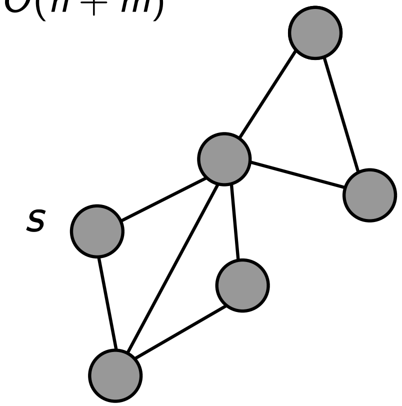
$O(nm)$



Kürzeste-Wege-Algorithmen

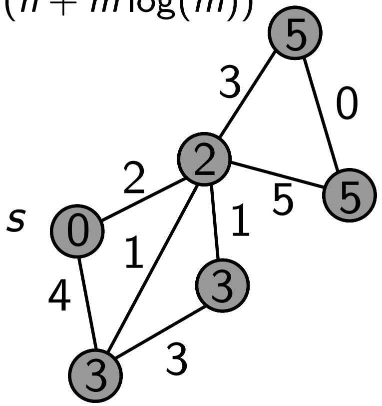
Breitensuche

$O(n+m)$



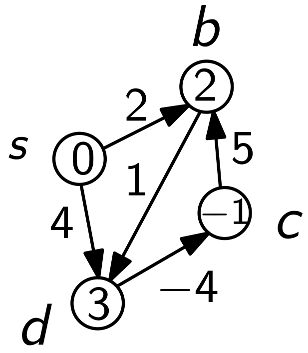
Dijkstra

$O(n+m\log(m))$



Bellman-Ford

$O(nm)$



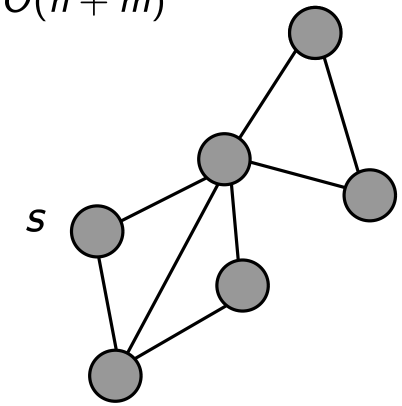
Floyd-Warshall

$O(n^3)$

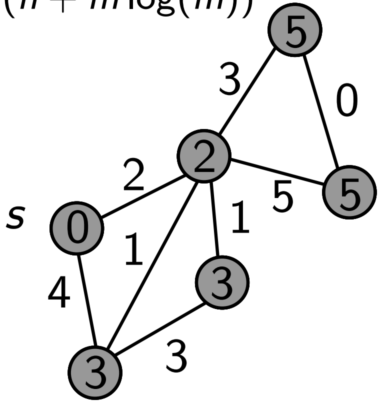
	s	b	c	d
s				
b				
c				
d				

Kürzeste-Wege-Algorithmen

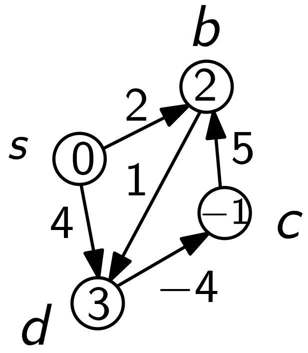
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



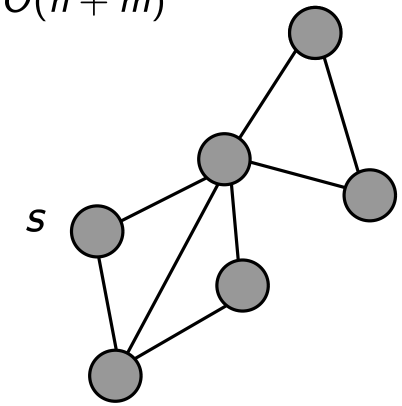
Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2		4
b		0		1
c		5	0	
d			-4	0

Kürzeste-Wege-Algorithmen

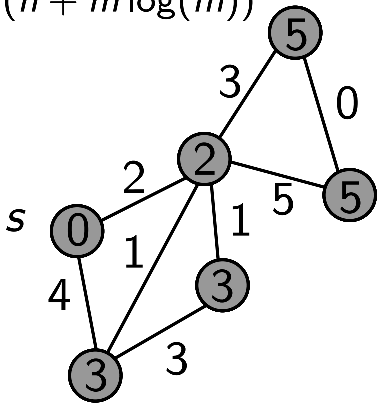
Breitensuche

$O(n + m)$



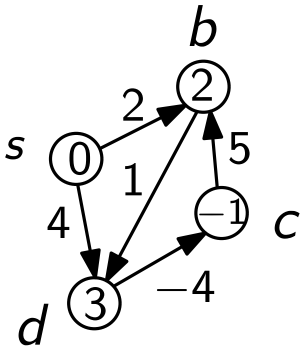
Dijkstra

$O(n + m\log(m))$



Bellman-Ford

$O(nm)$



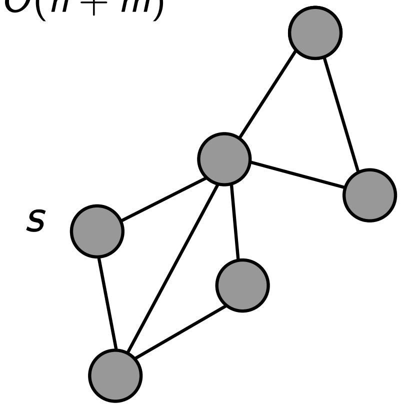
Floyd-Warshall

$O(n^3)$

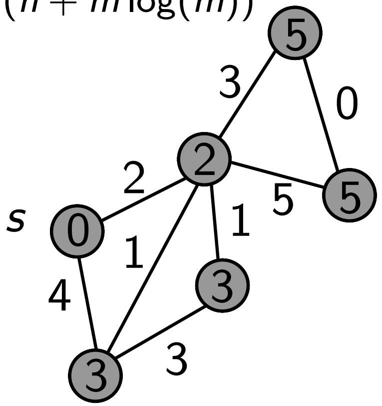
	s	b	c	d
s	0	2		4
b		0		1
c		5	0	
d			-4	0

Kürzeste-Wege-Algorithmen

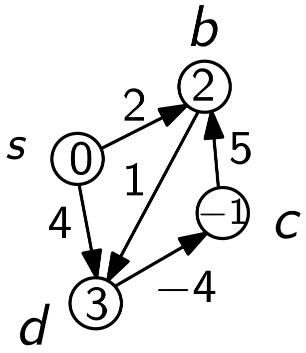
Breitensuche
$O(n + m)$



Dijkstra
$O(n + m\log(m))$



Bellman-Ford
$O(nm)$

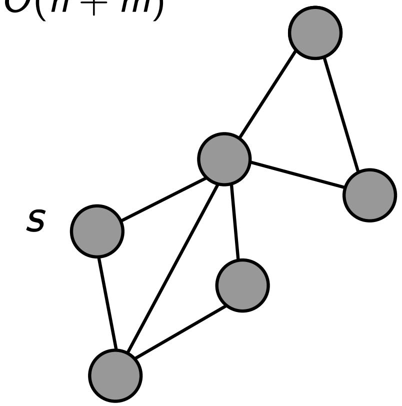


Floyd-Warshall
$O(n^3)$

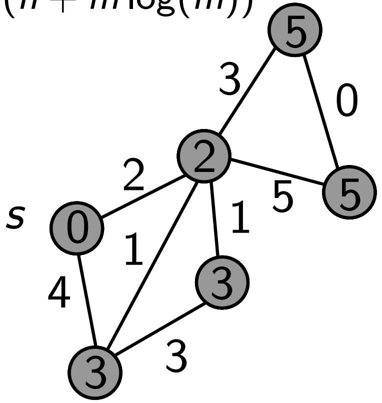
	s	b	c	d
s	0	2		3
b		0		1
c		5	0	6
d			−4	0

Kürzeste-Wege-Algorithmen

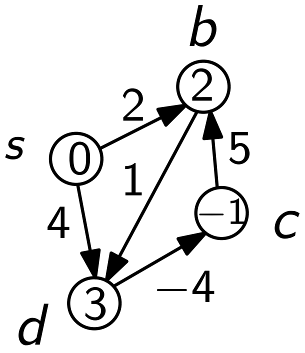
Breitensuche
$O(n + m)$



Dijkstra
$O(n + m \log(m))$



Bellman-Ford
$O(nm)$

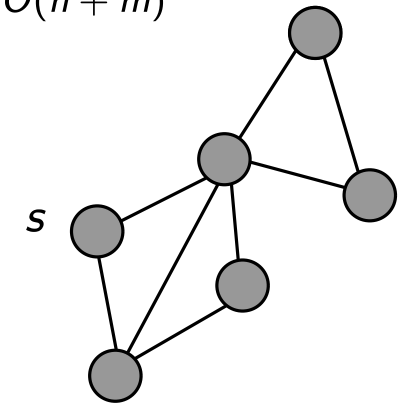


Floyd-Warshall
$O(n^3)$

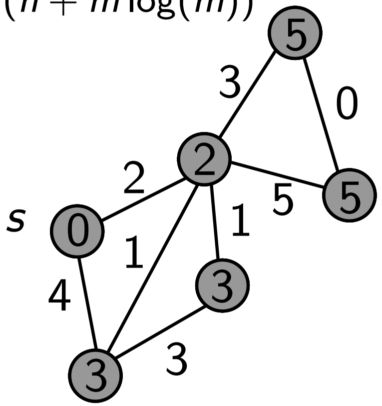
	s	b	c	d
s	0	2		3
b		0		1
c		5	0	6
d		1	-4	0

Kürzeste-Wege-Algorithmen

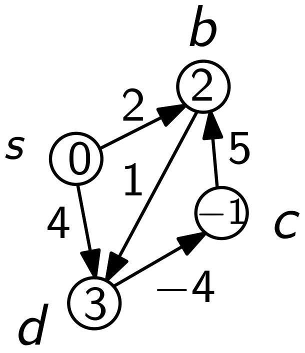
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$

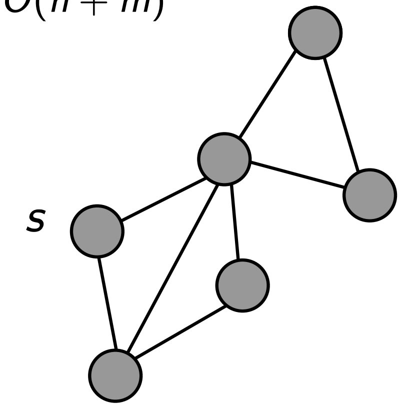


Floyd-Warshall
$O(n^3)$

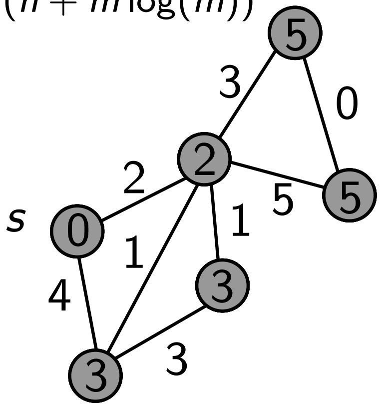
	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

Kürzeste-Wege-Algorithmen

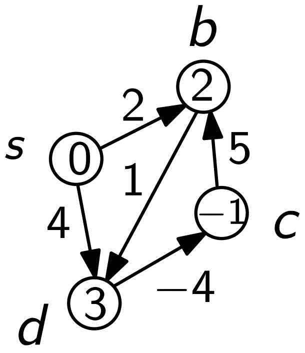
Breitensuche
$O(n+m)$



Dijkstra
$O(n + m\log(m))$



Bellman-Ford
$O(nm)$



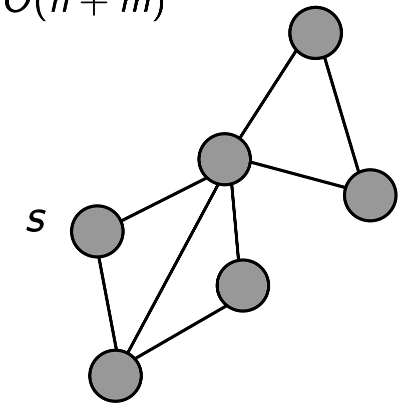
Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

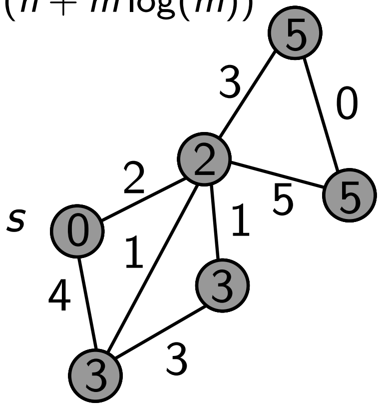
Welcher Algorithmus ist am besten geeignet?

Kürzeste-Wege-Algorithmen

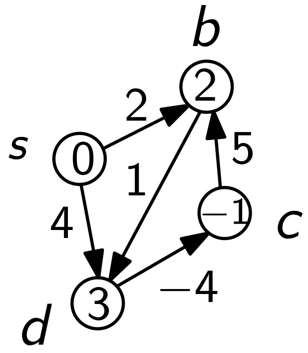
Breitensuche
$O(n + m)$



Dijkstra
$O(n + m \log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

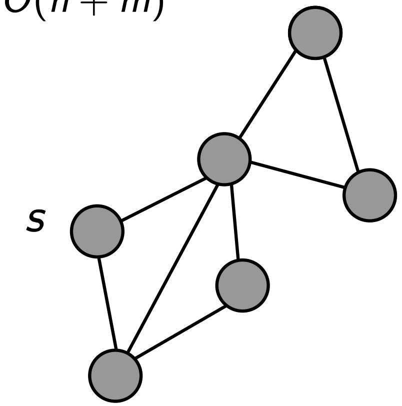
	s	b	c	d
s	0	2	-1	3
b		0	-3	1
c		5	0	6
d		1	-4	0

Welcher Algorithmus ist am besten geeignet?

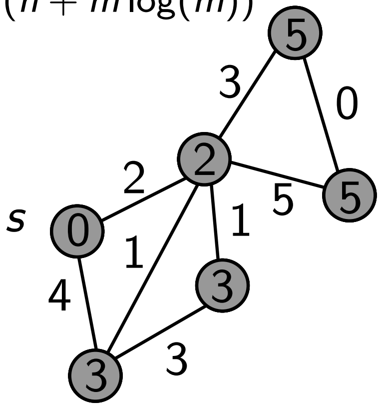
- Startknoten s, alle Gewichte sind positiv

Kürzeste-Wege-Algorithmen

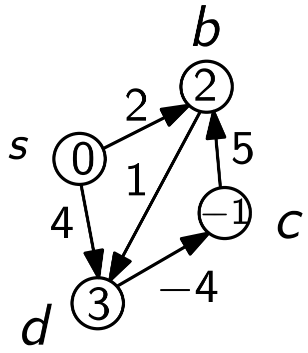
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

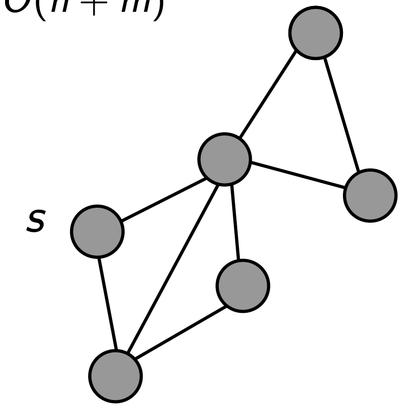
	s	b	c	d
s	0	2	-1	3
b		0	-3	1
c		5	0	6
d		1	-4	0

Welcher Algorithmus ist am besten geeignet?

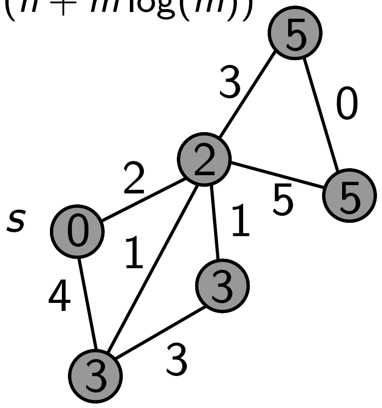
- Startknoten s, alle Gewichte sind positiv
- Startknoten s, alle Gewichte sind gleich und positiv

Kürzeste-Wege-Algorithmen

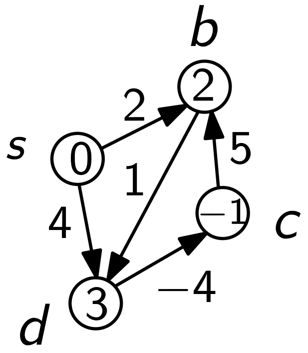
Breitensuche
$O(n + m)$



Dijkstra
$O(n + m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

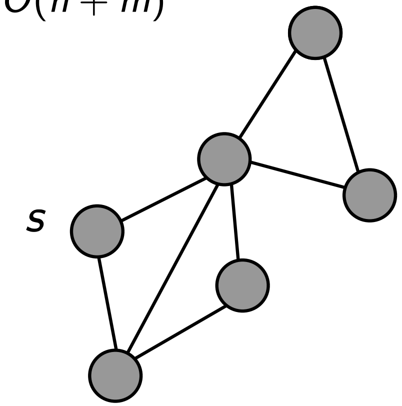
	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

Welcher Algorithmus ist am besten geeignet?

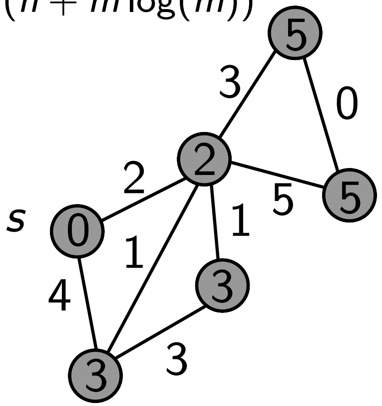
- Startknoten s, alle Gewichte sind positiv
- Startknoten s, alle Gewichte sind gleich und positiv
- APSP, negative Gewichte

Kürzeste-Wege-Algorithmen

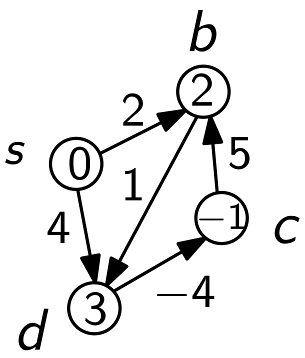
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	-1	3
b		0	-3	1
c		5	0	6
d		1	-4	0

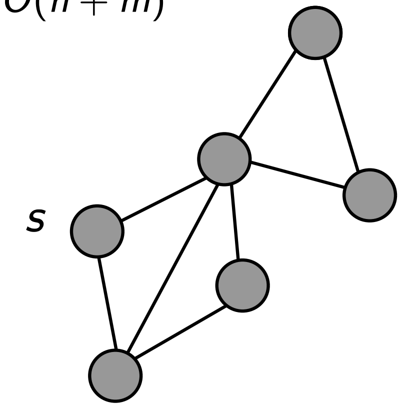
Welcher Algorithmus ist am besten geeignet?

- Startknoten s, alle Gewichte sind positiv
- Startknoten s, alle Gewichte sind gleich und positiv
- APSP, negative Gewichte
- APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)

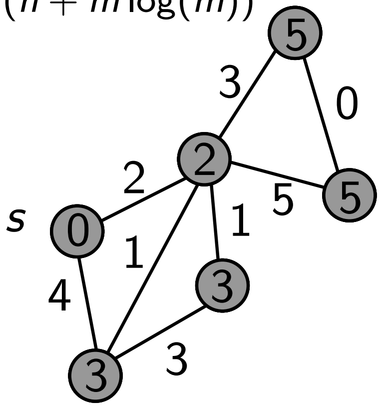
pwa.klicker.uzh.ch/join/algo1

Kürzeste-Wege-Algorithmen

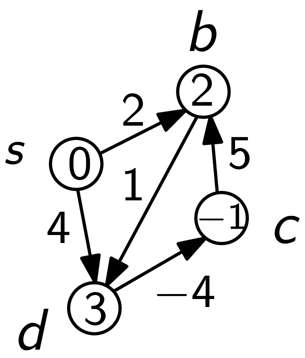
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	-1	3
b		0	-3	1
c		5	0	6
d		1	-4	0

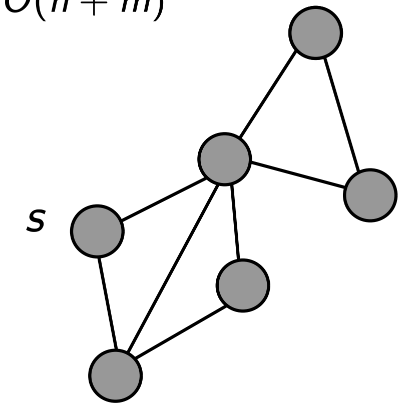
BFS **Dijkstra** **B-F** **F-W**

Welcher Algorithmus ist am besten geeignet?

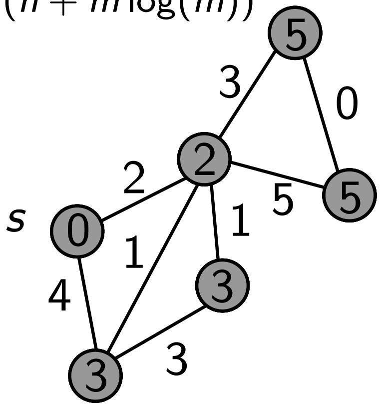
- Startknoten s, alle Gewichte sind positiv
- Startknoten s, alle Gewichte sind gleich und positiv
- APSP, negative Gewichte
- APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)

Kürzeste-Wege-Algorithmen

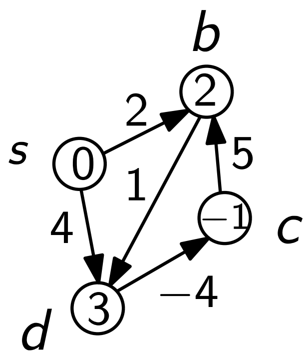
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	-1	3
b		0	-3	1
c		5	0	6
d		1	-4	0

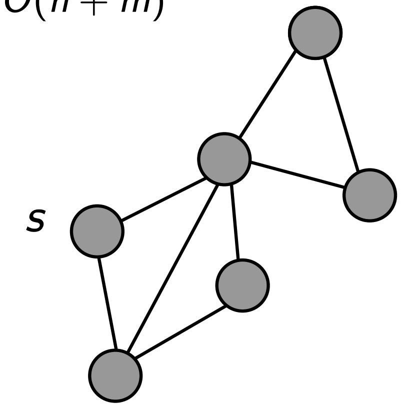
Welcher Algorithmus ist am besten geeignet?

- Startknoten s, alle Gewichte sind positiv
- Startknoten s, alle Gewichte sind gleich und positiv
- APSP, negative Gewichte
- APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)

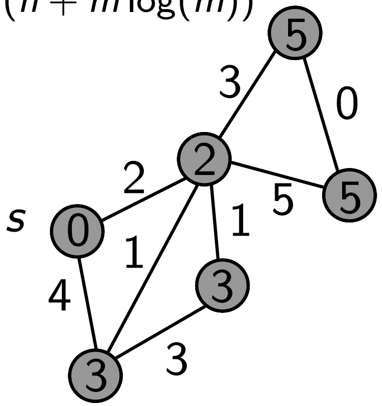
BFS	Dijkstra	B-F	F-W
✗	✓	✓	✓

Kürzeste-Wege-Algorithmen

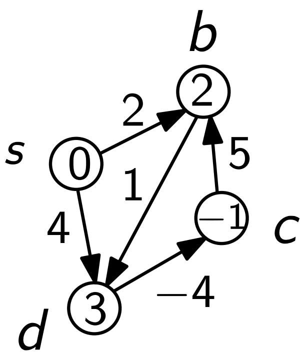
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

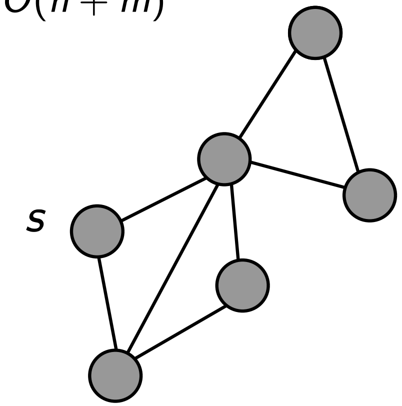
Welcher Algorithmus ist am besten geeignet?

- Startknoten s, alle Gewichte sind positiv
- Startknoten s, alle Gewichte sind gleich und positiv
- APSP, negative Gewichte
- APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)

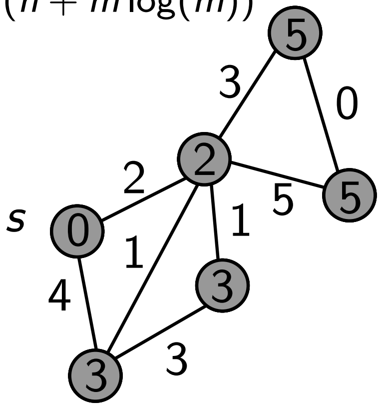
BFS	Dijkstra	B-F	F-W
✗	✓✓	✓	✓

Kürzeste-Wege-Algorithmen

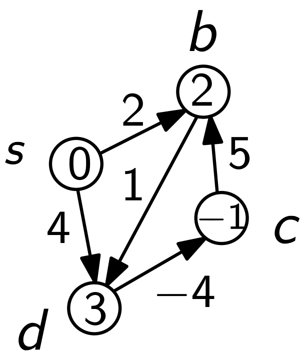
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

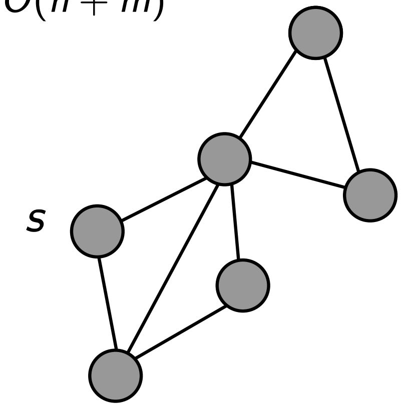
Welcher Algorithmus ist am besten geeignet?

	BFS	Dijkstra	B-F	F-W
Startknoten s, alle Gewichte sind positiv	✗	✓✓	✓	✓
Startknoten s, alle Gewichte sind gleich und positiv		✓	✓	✓
APSP, negative Gewichte				
APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)				

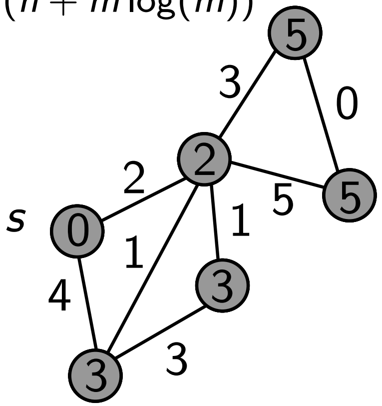
Kürzeste-Wege-Algorithmen

Karlsruher Institut für Technologie

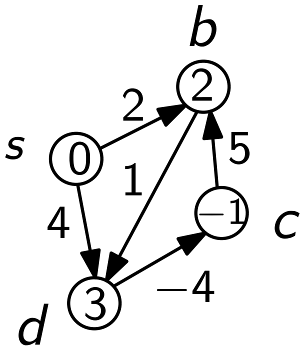
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	-1	3
b		0	-3	1
c		5	0	6
d		1	-4	0

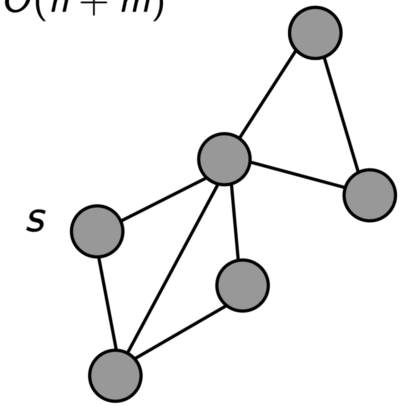
Welcher Algorithmus ist am besten geeignet?

- Startknoten s, alle Gewichte sind positiv
- Startknoten s, alle Gewichte sind gleich und positiv
- APSP, negative Gewichte
- APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)

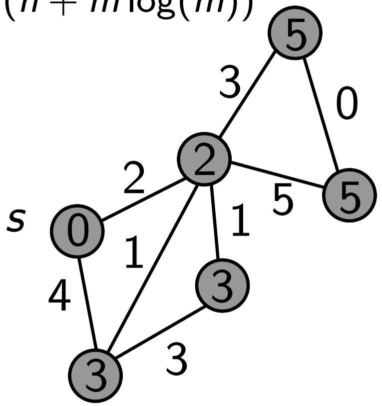
	BFS	Dijkstra	B-F	F-W
Startknoten s, alle Gewichte sind positiv	✗	✓✓	✓	✓
Startknoten s, alle Gewichte sind gleich und positiv	✓✓	✓	✓	✓
APSP, negative Gewichte				
APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)				

Kürzeste-Wege-Algorithmen

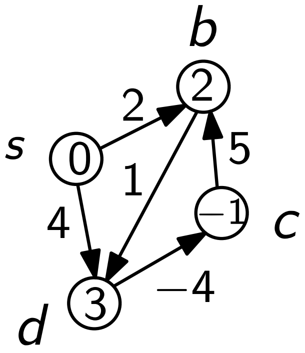
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



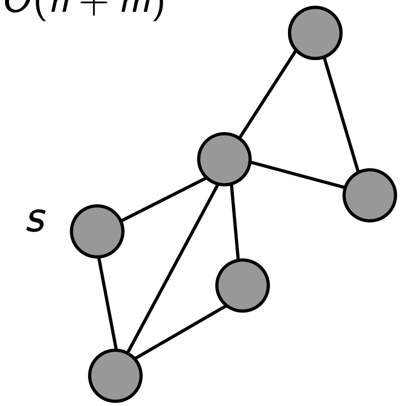
Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

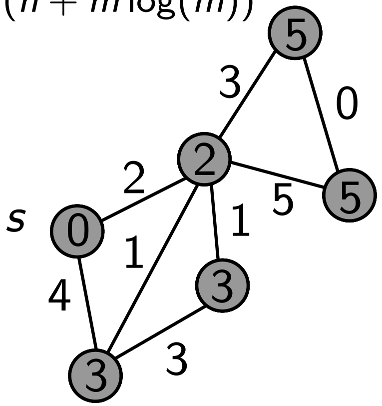
Welcher Algorithmus ist am besten geeignet?	**BFS**	**Dijkstra**	**B-F**	**F-W**
Startknoten s, alle Gewichte sind positiv	✗	✓✓	✓	✓
Startknoten s, alle Gewichte sind gleich und positiv	✓✓	✓	✓	✓
APSP, negative Gewichte	✗	✗	✓	✓
APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)				

Kürzeste-Wege-Algorithmen

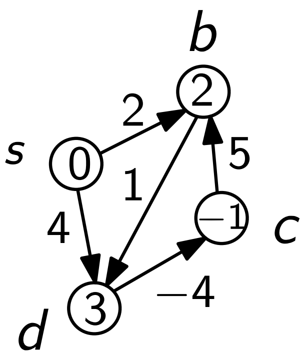
Breitensuche
$O(n + m)$



Dijkstra
$O(n + m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

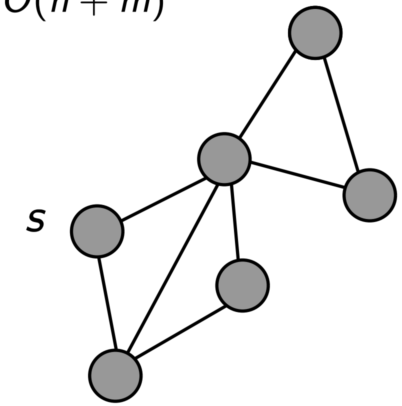
	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

Welcher Algorithmus ist am besten geeignet?

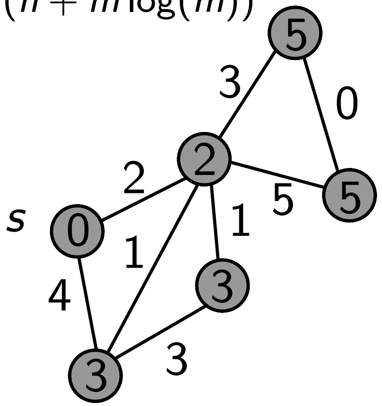
	BFS	Dijkstra	B-F	F-W
Startknoten s, alle Gewichte sind positiv	✗	✓✓	✓	✓
Startknoten s, alle Gewichte sind gleich und positiv	✓✓	✓	✓	✓
APSP, negative Gewichte	✗	✗	✓	✓✓
APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)				

Kürzeste-Wege-Algorithmen

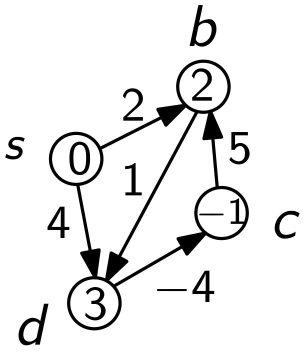
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



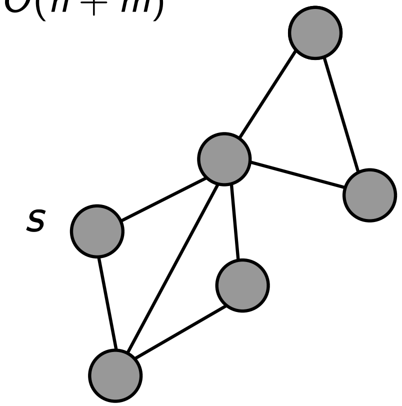
Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

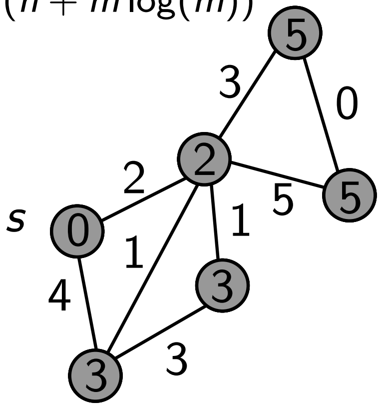
Welcher Algorithmus ist am besten geeignet?	**BFS**	**Dijkstra**	**B-F**	**F-W**
Startknoten s, alle Gewichte sind positiv	✗	✓✓	✓	✓
Startknoten s, alle Gewichte sind gleich und positiv	✓✓	✓	✓	✓
APSP, negative Gewichte	✗	✗	✓	✓✓
APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)	✗	✓	✓	✓

Kürzeste-Wege-Algorithmen

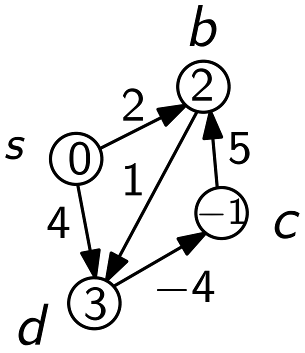
Breitensuche
$O(n+m)$



Dijkstra
$O(n+m\log(m))$



Bellman-Ford
$O(nm)$



Floyd-Warshall
$O(n^3)$

	s	b	c	d
s	0	2	−1	3
b		0	−3	1
c		5	0	6
d		1	−4	0

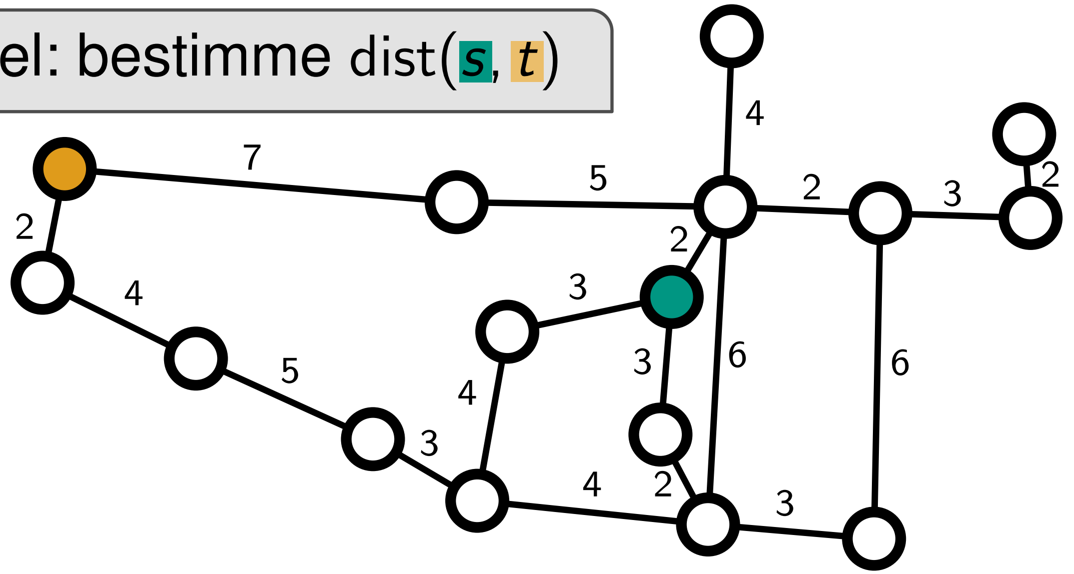
Welcher Algorithmus ist am besten geeignet?	**BFS**	**Dijkstra**	**B-F**	**F-W**
Startknoten s, alle Gewichte sind positiv	✗	✓✓	✓	✓
Startknoten s, alle Gewichte sind gleich und positiv	✓✓	✓	✓	✓
APSP, negative Gewichte	✗	✗	✓	✓✓
APSP, alle Gewichte sind positiv, $m \in O(n)$ (wenig Kanten)	✗	✓✓	✓	✓

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**$(v, d[v])$

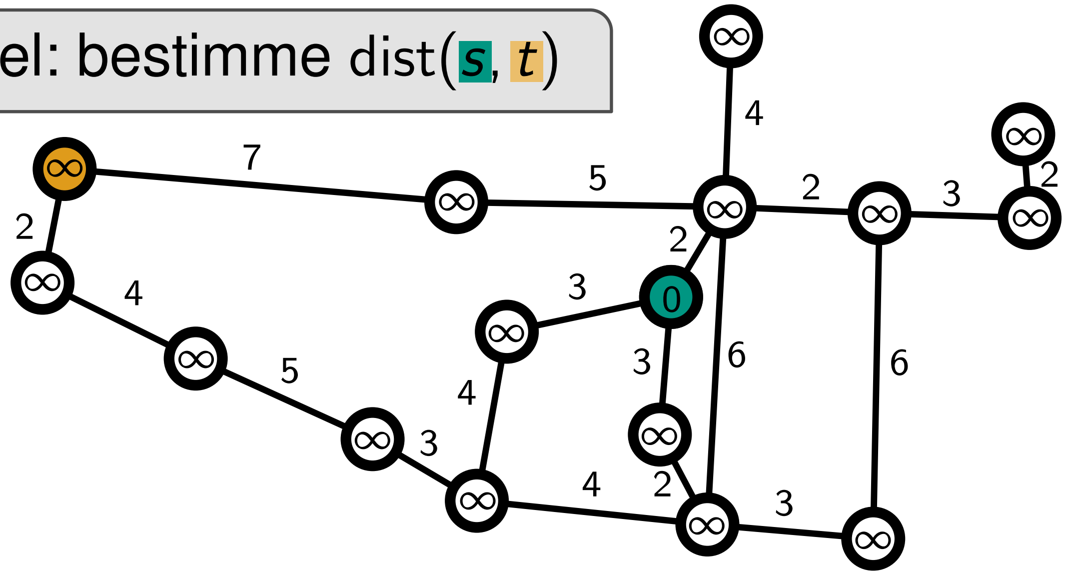
Ziel: bestimme dist(s, t)



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**$(v, d[v])$

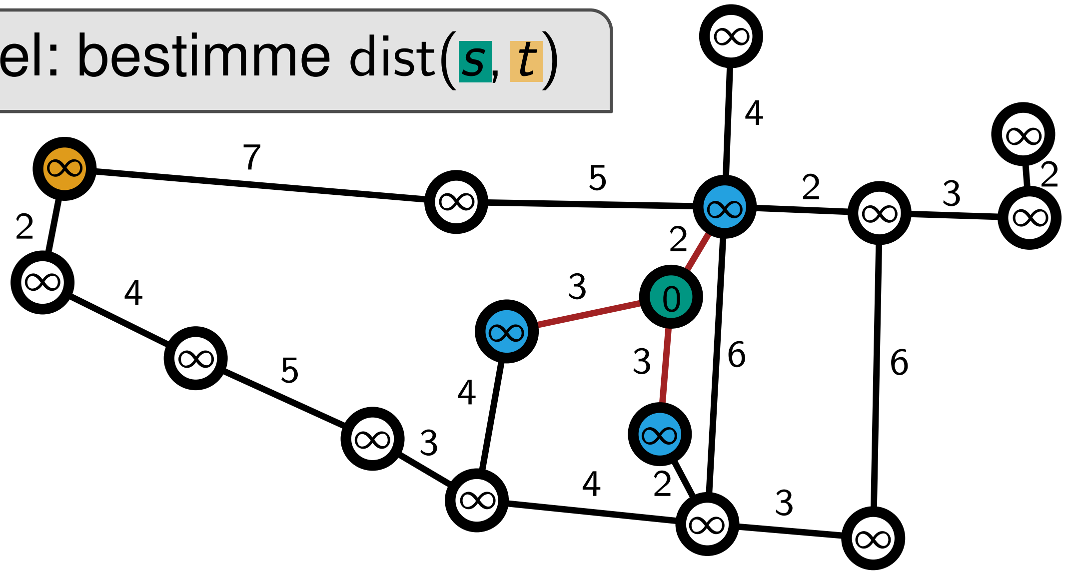
Ziel: bestimme dist(s, t)



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 dist(s, t)

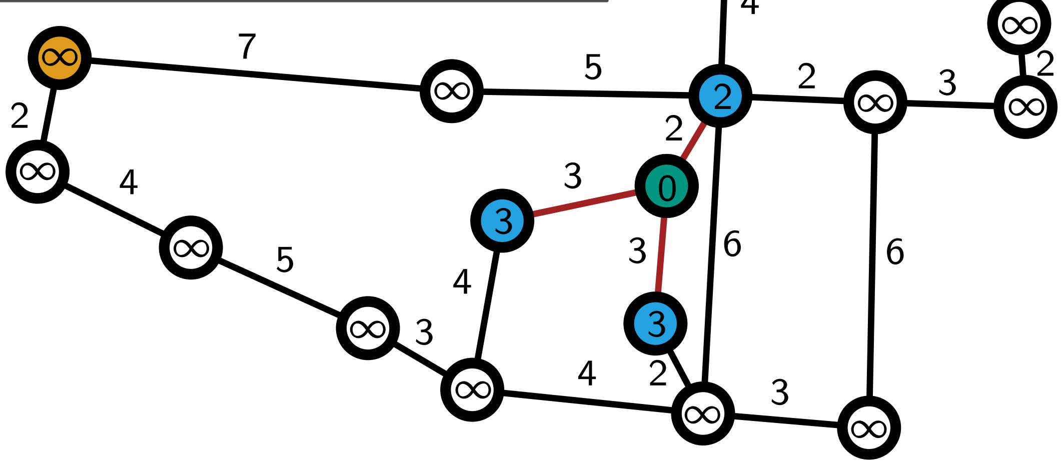


Dijkstra (Wiederholung)

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 dist(s, t)

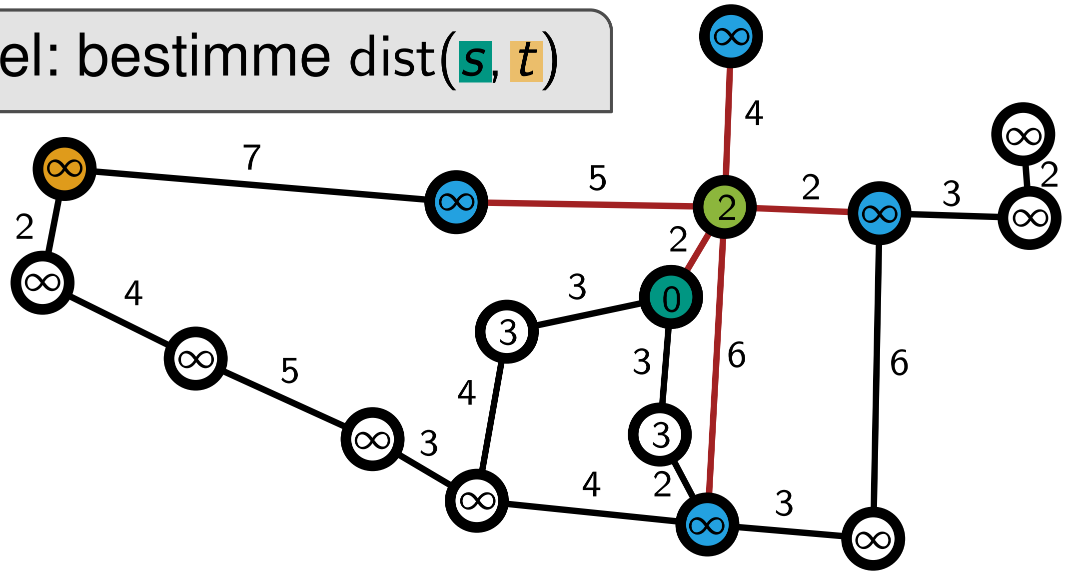


Dijkstra (Wiederholung)

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 dist(s, t)



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**$()$

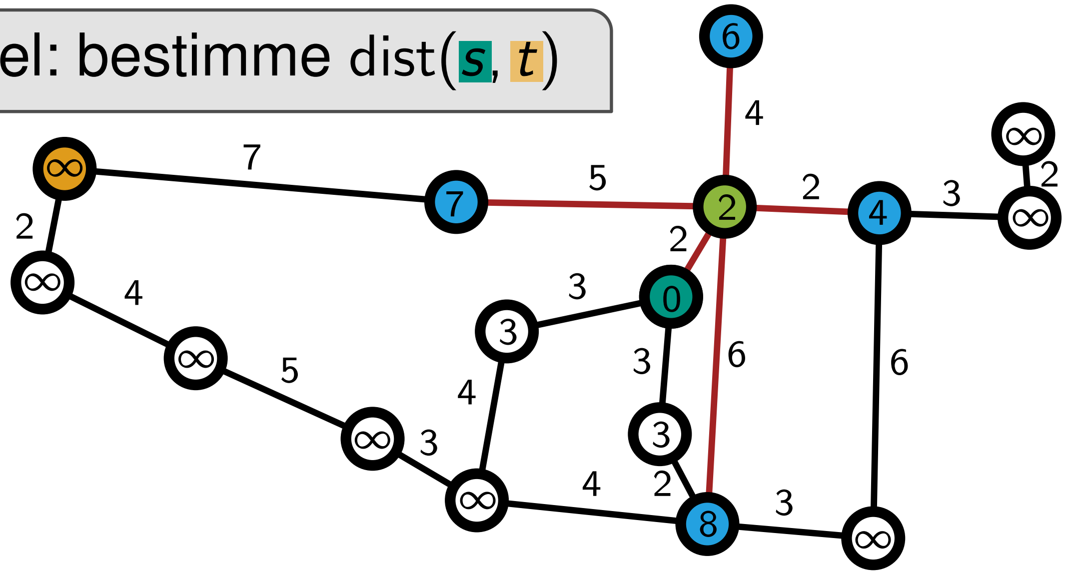
  for *Node* v in $N(u)$ **do**

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

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

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

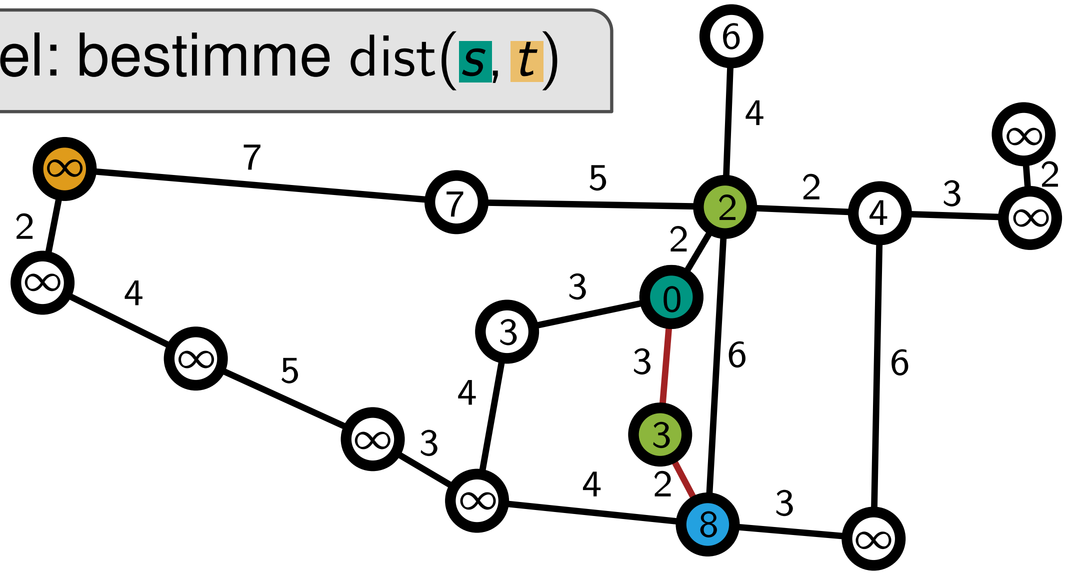
Ziel: bestimme dist(s, t)



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])
```

Ziel: bestimme dist(s, t)



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**$()$

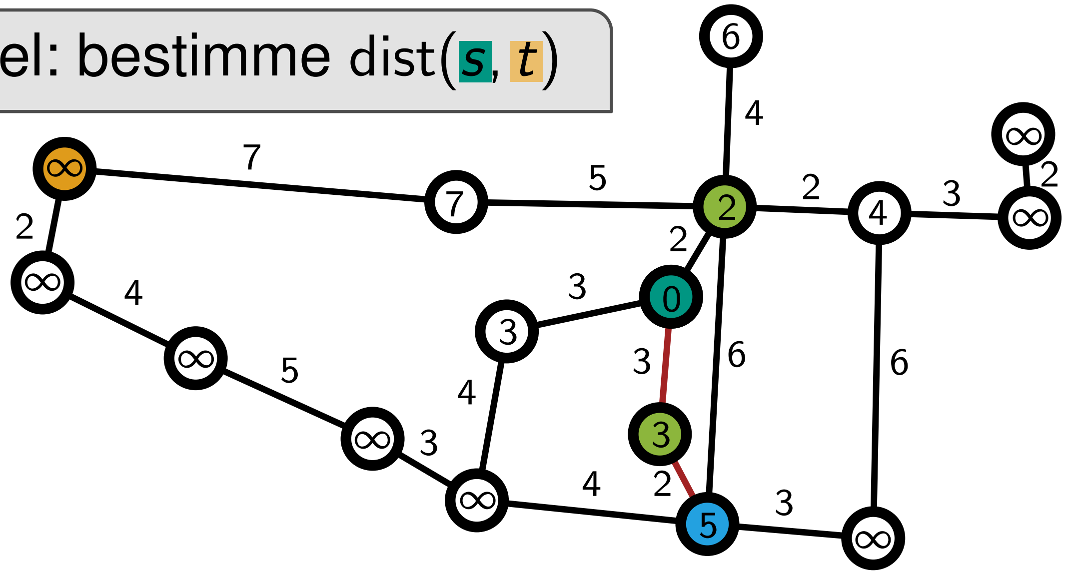
  for *Node* v in $N(u)$ **do**

    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 dist(s, t)

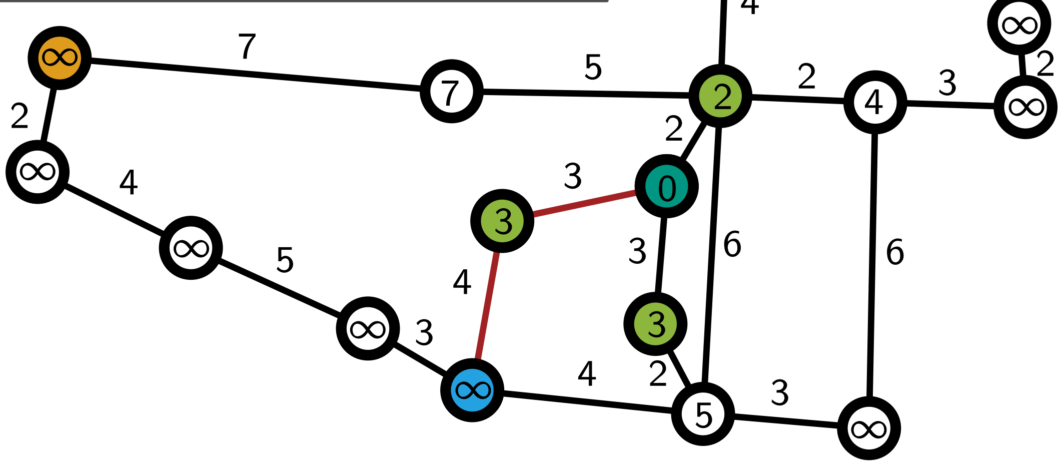


Dijkstra (Wiederholung)

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 dist(s, t)

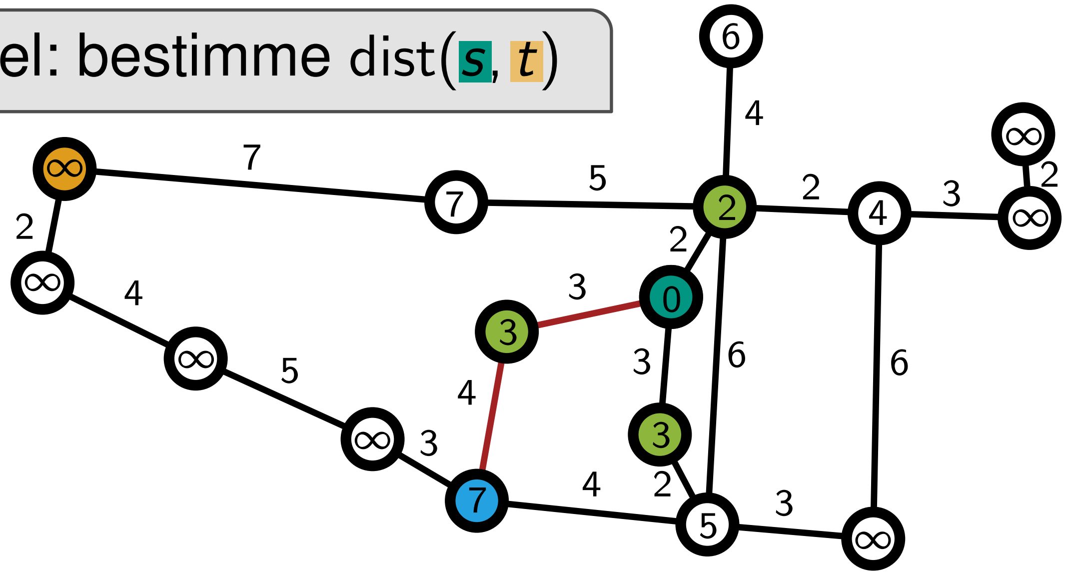


Dijkstra (Wiederholung)

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 dist(s, t)



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**()

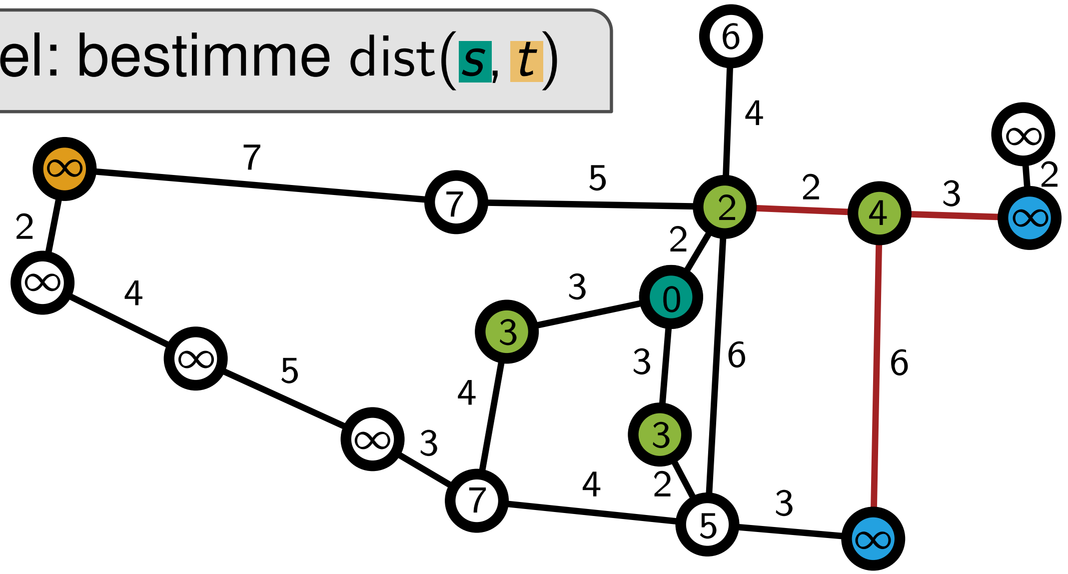
  for *Node* v in $N(u)$ **do**

    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 dist(s, t)



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**$()$

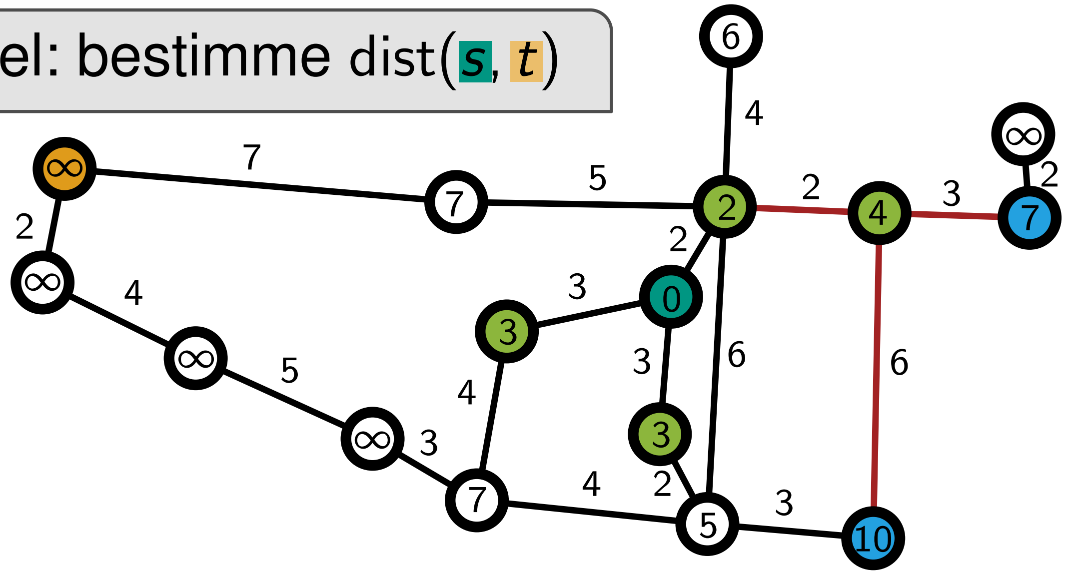
  for $Node\ v$ in $N(u)$ **do**

    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 dist(s, t)

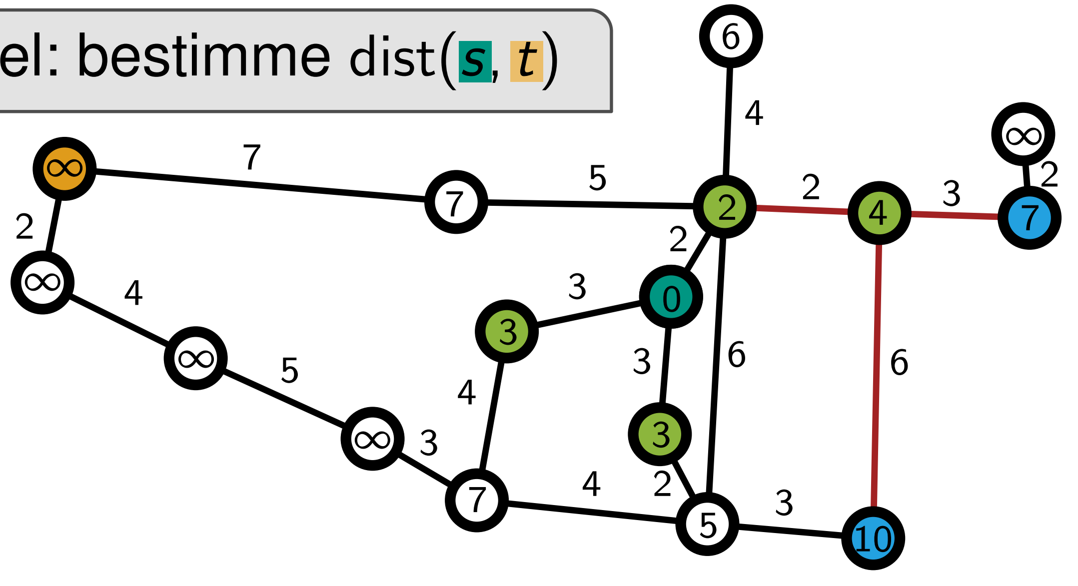


Dijkstra (Wiederholung)

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
    if $d[v] > d[$
      $d[v] := d$
      Q.**decPr**

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



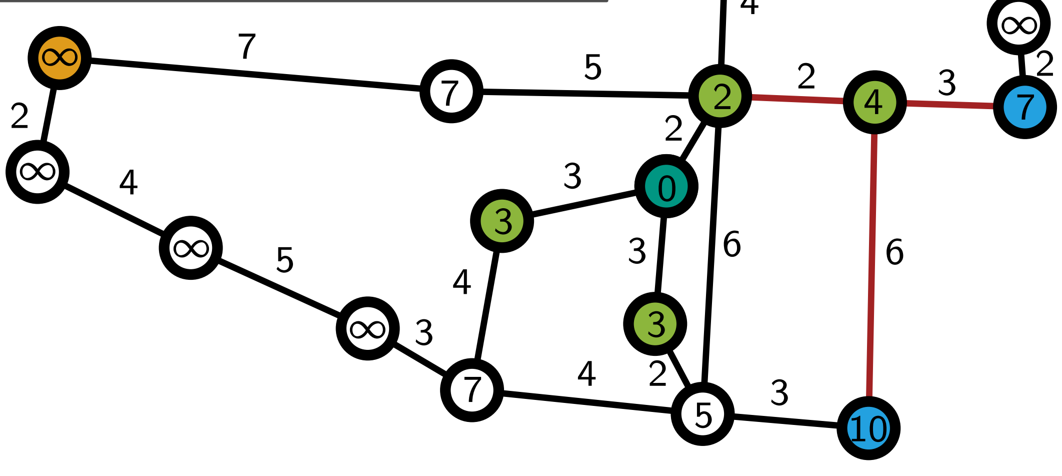
Wann ist $\text{dist}(s, t)$ bekannt?

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**

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



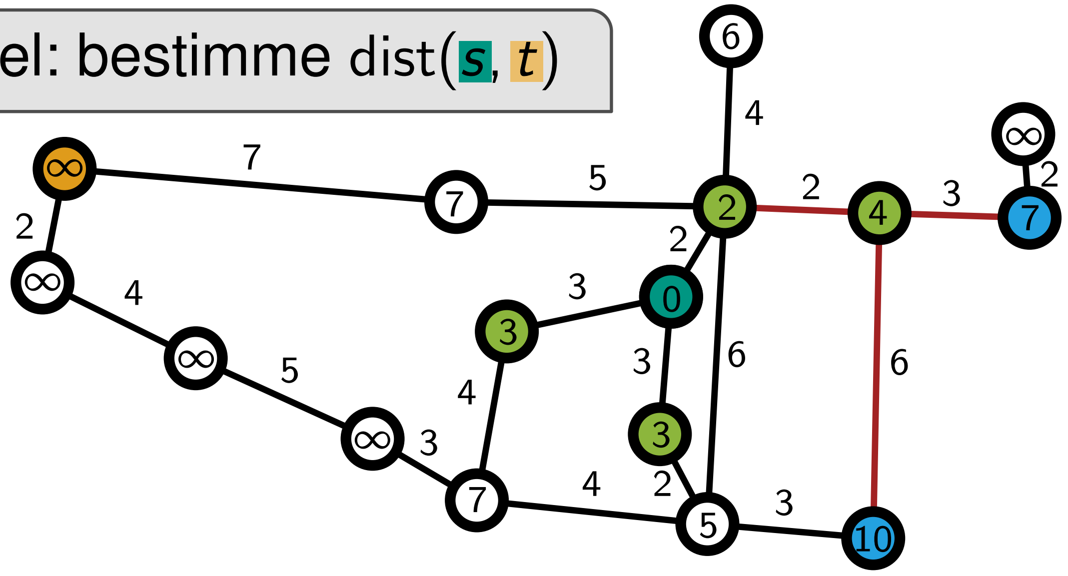
Wann ist $\text{dist}(s, t)$ bekannt?

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 dist(s, t)



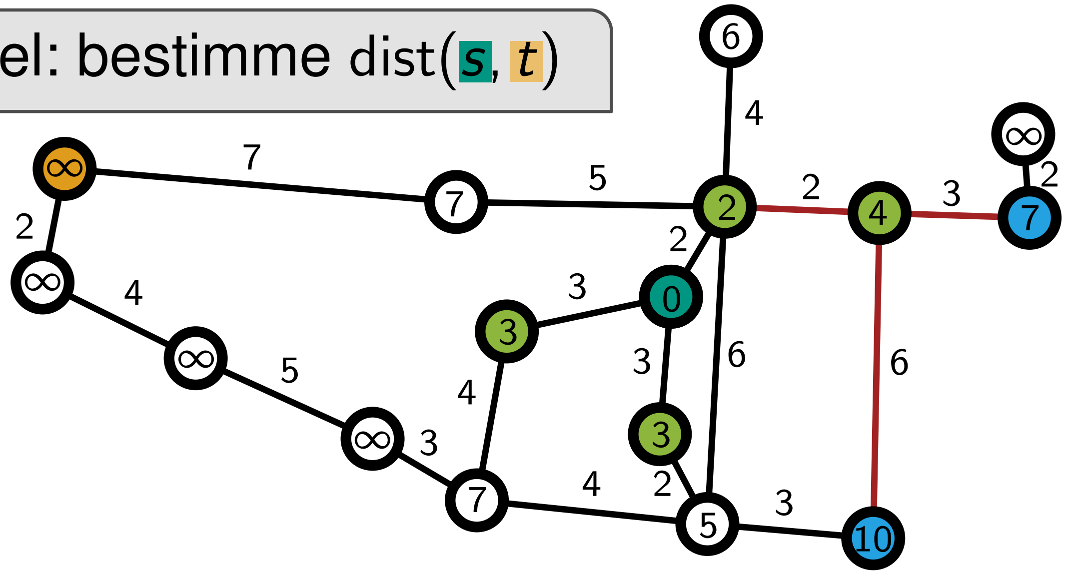
Wie viele Knoten werden aus der Queue entfernt, bevor t entfernt wird?

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?

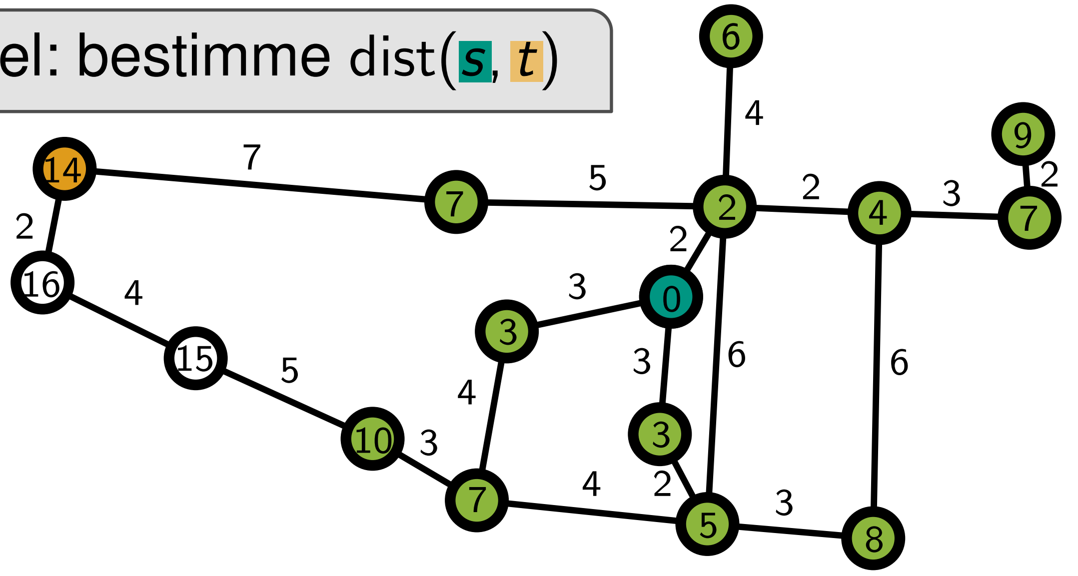
- Knoten werden mit nicht-absteigender Distanz zu s aus der Queue entfernt
- alle Knoten v mit $\text{dist}(s, v) < \text{dist}(s, t)$ werden vor t entfernt

Dijkstra (Wiederholung)

$\textbf{Dijkstra}(\textit{Graph } G, \textit{Node } s)$

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

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



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

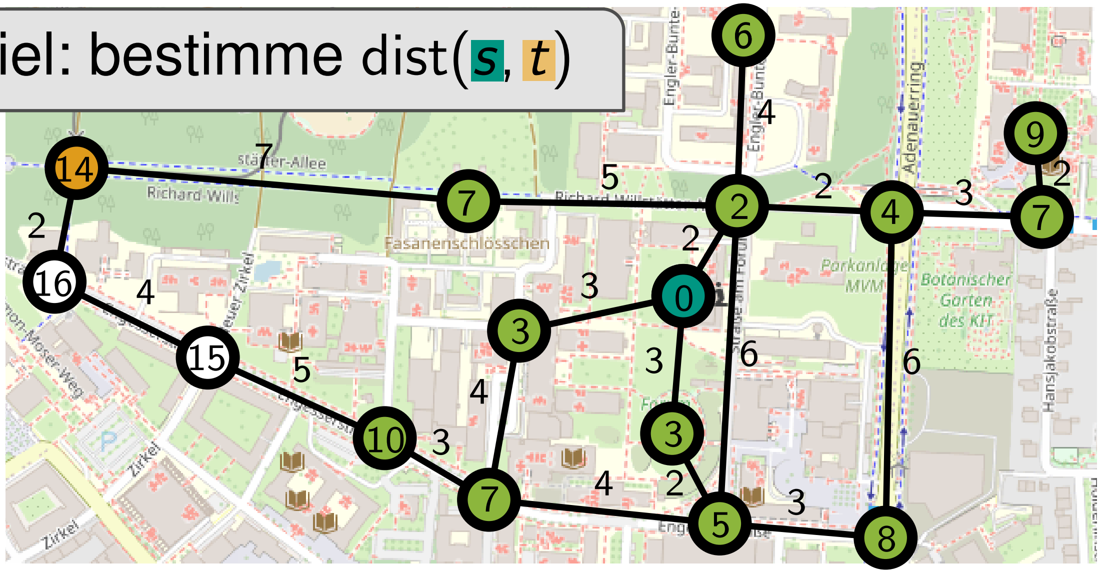
- Knoten werden mit nicht-absteigender Distanz zu s aus der Queue entfernt
- alle Knoten v mit $\text{dist}(s, v) < \text{dist}(s, t)$ werden vor t entfernt

Dijkstra (Wiederholung)

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)



Neues Setting

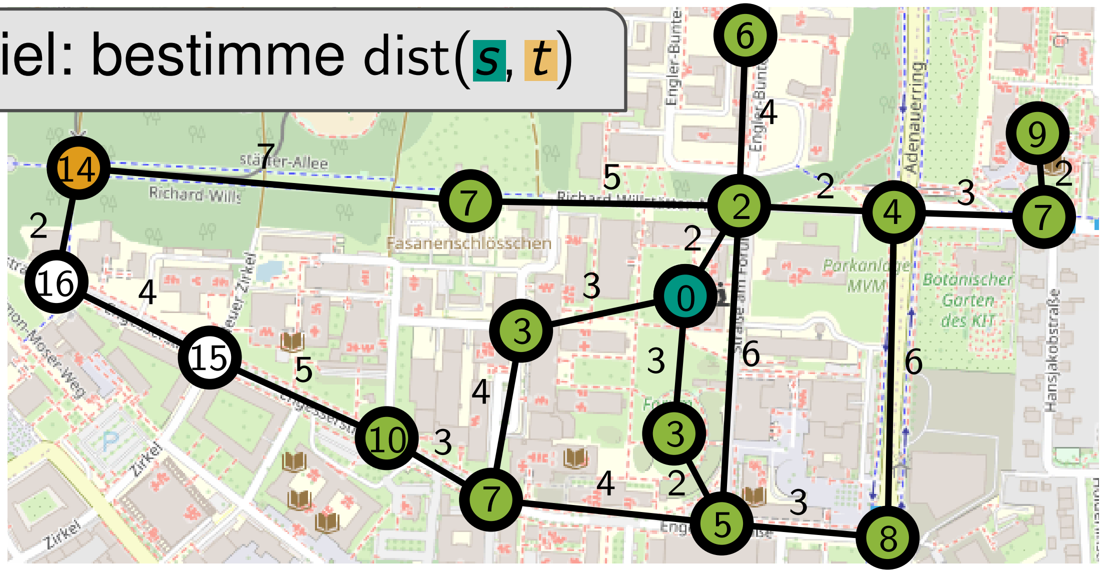
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)

Karlsruher Institut für Technologie

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 dist(s, t)



Neues Setting

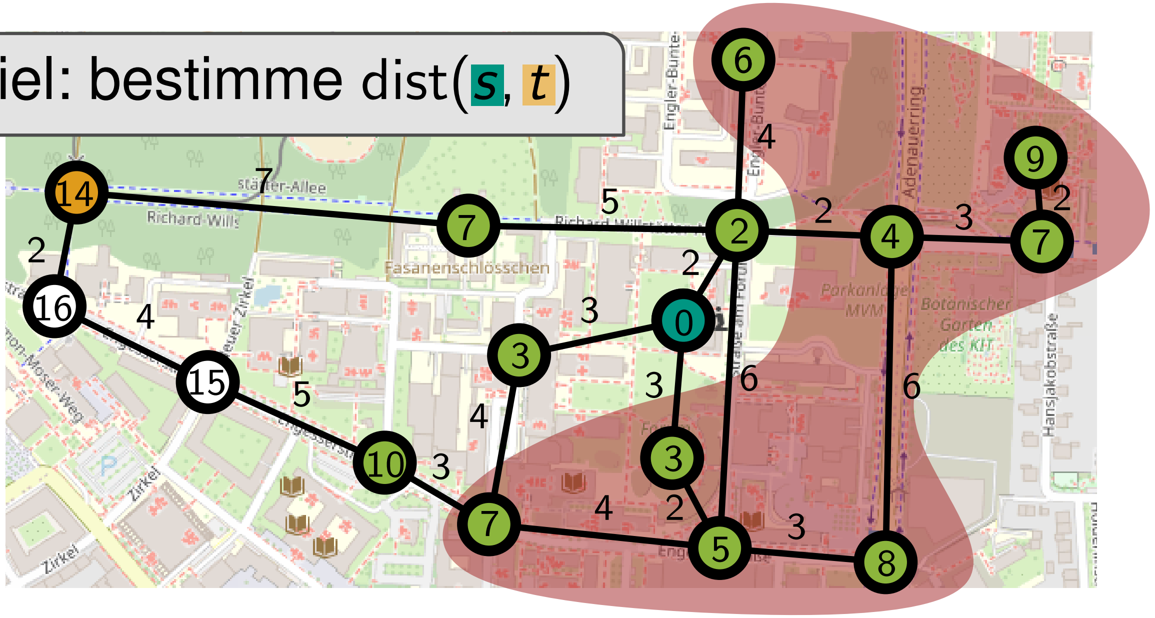
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \hat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

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**$(v, d[v])$

Ziel: bestimme dist(s, t)

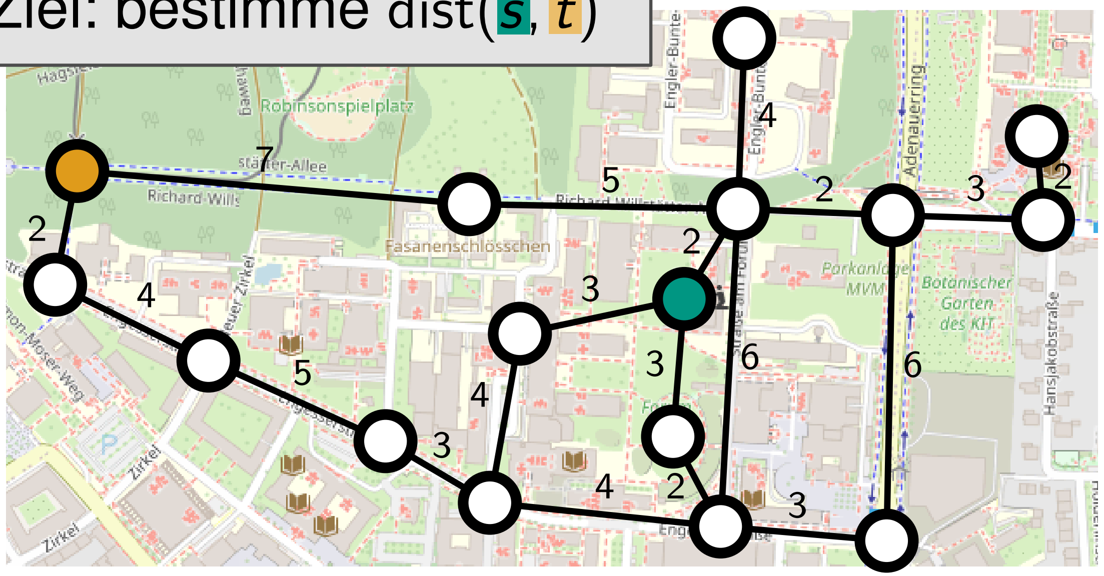


Neues Setting

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)

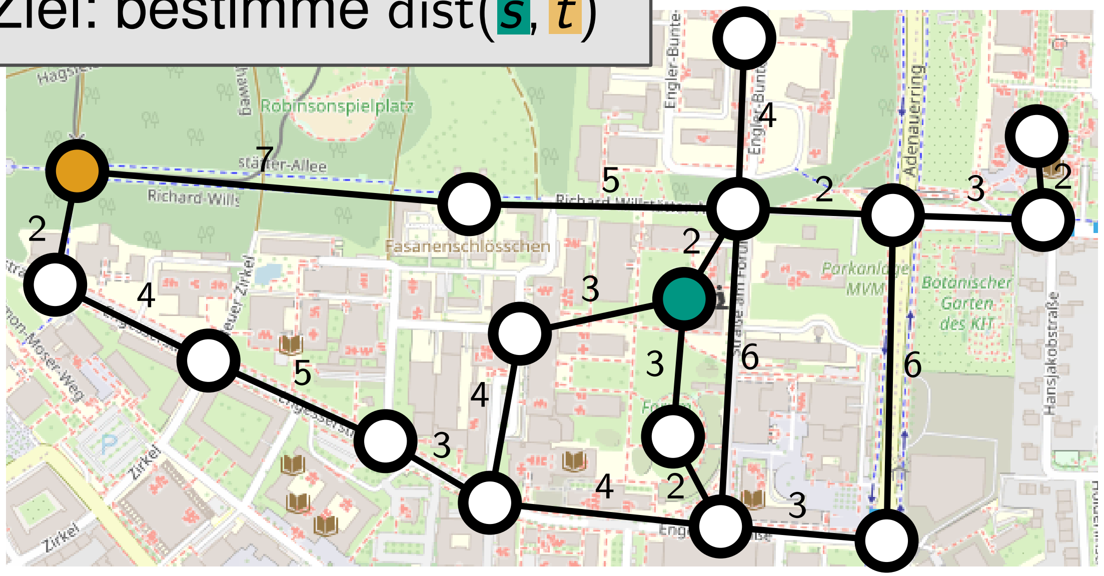


Neues Setting

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Neues Setting

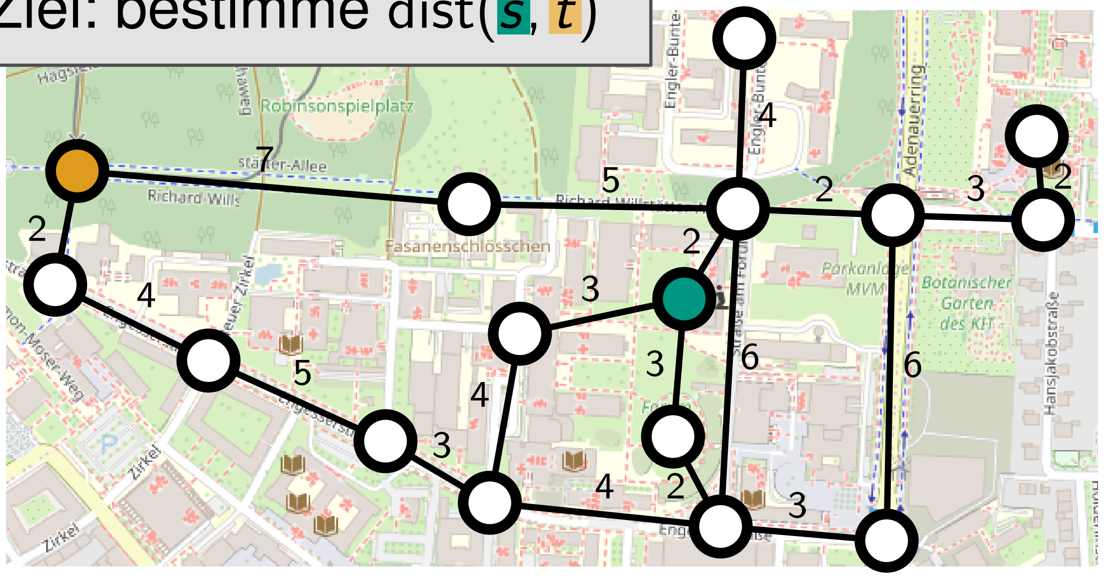
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

- mache gute Kanten günstiger und schlechte Kanten teurer

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Idee

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

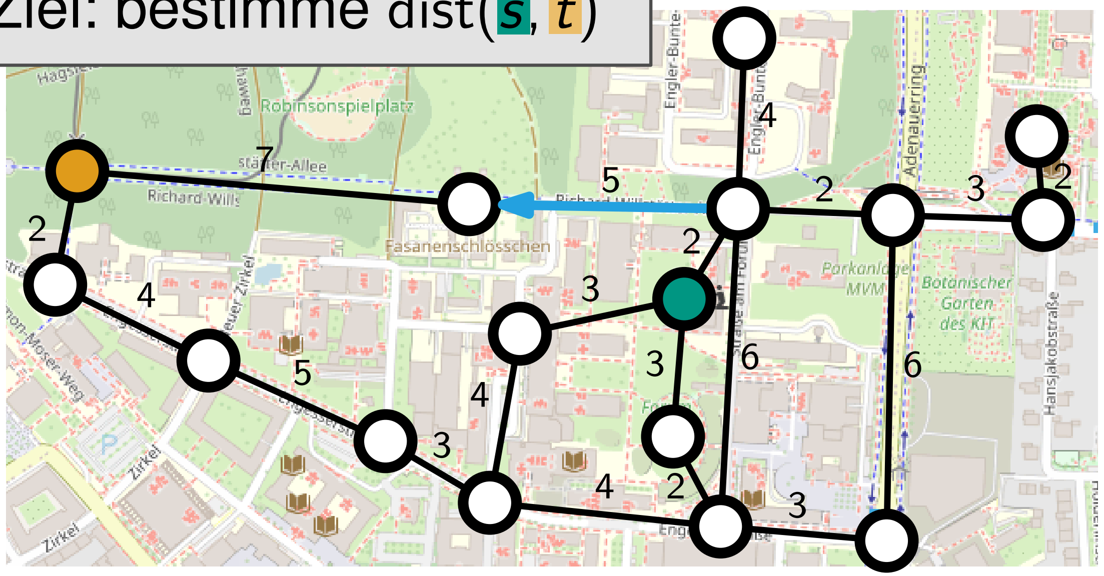
Neues Setting

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

Dijkstra (zielgerichtet)

Karlsruher Institut für Technologie

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



Neues Setting

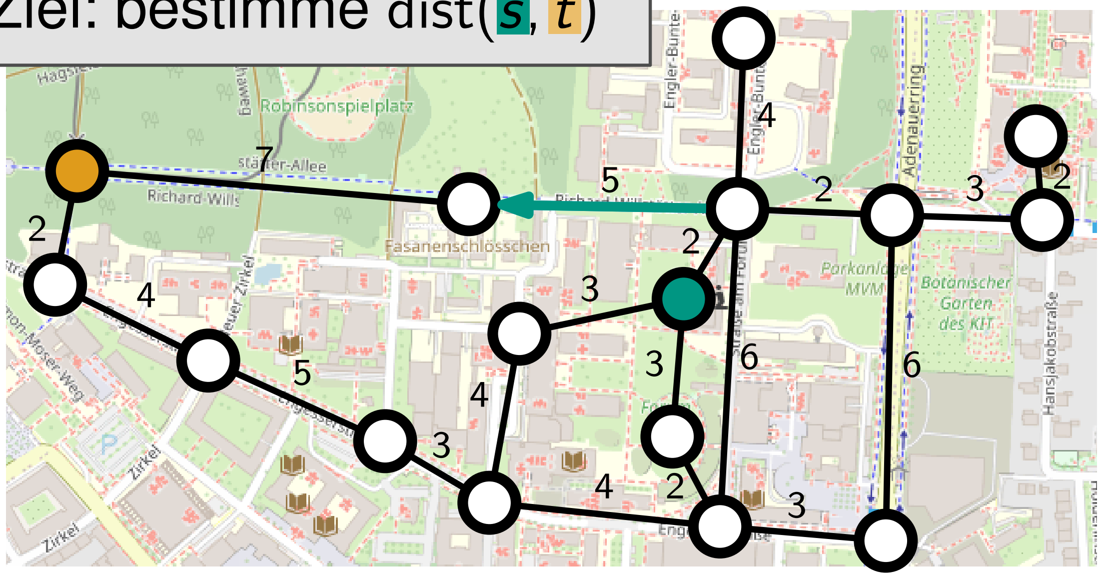
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Neues Setting

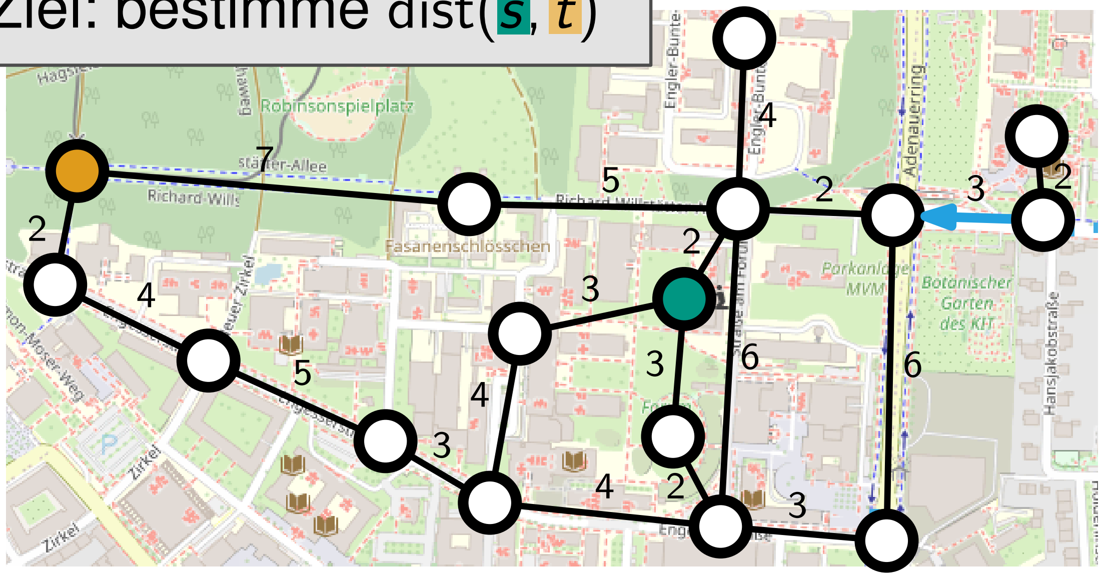
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Neues Setting

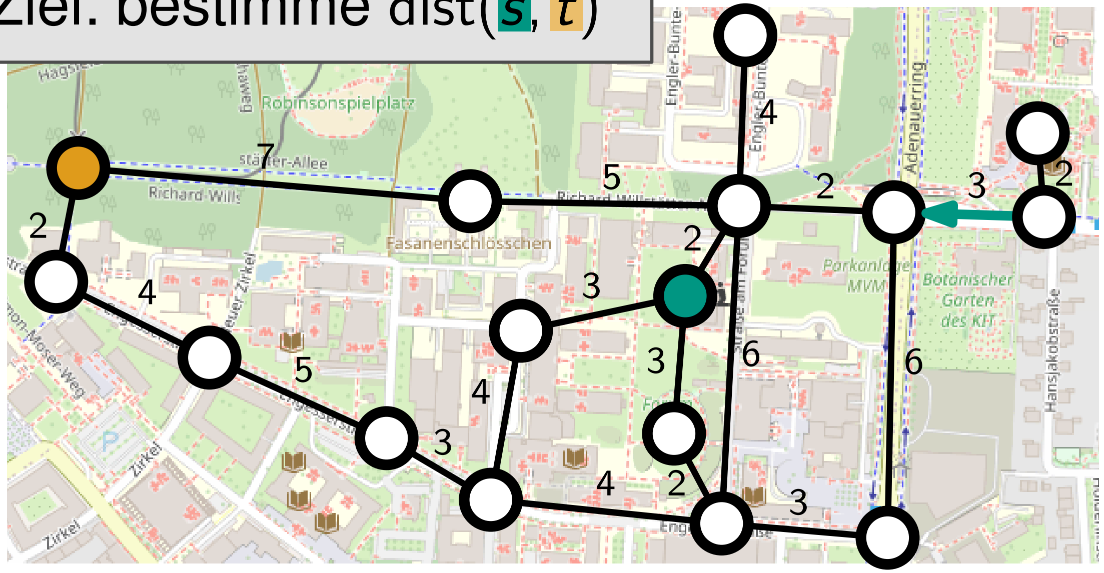
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Neues Setting

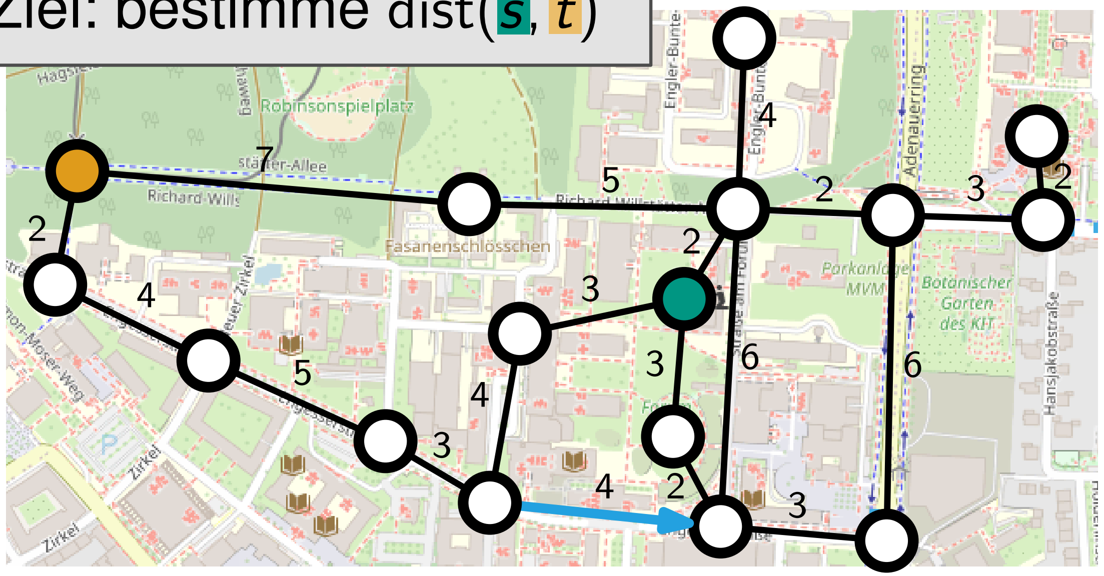
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Neues Setting

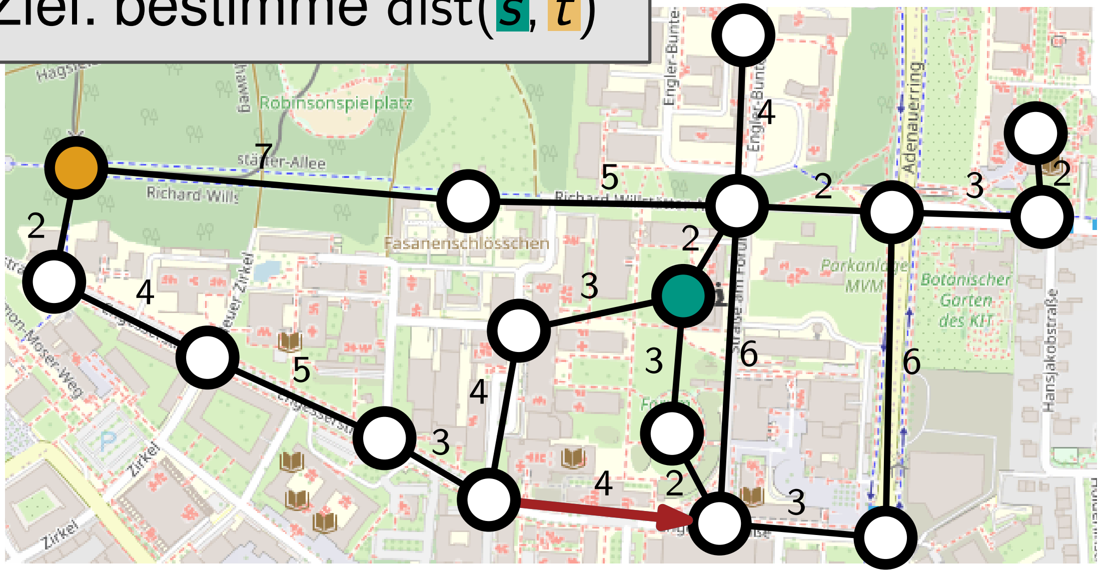
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Idee

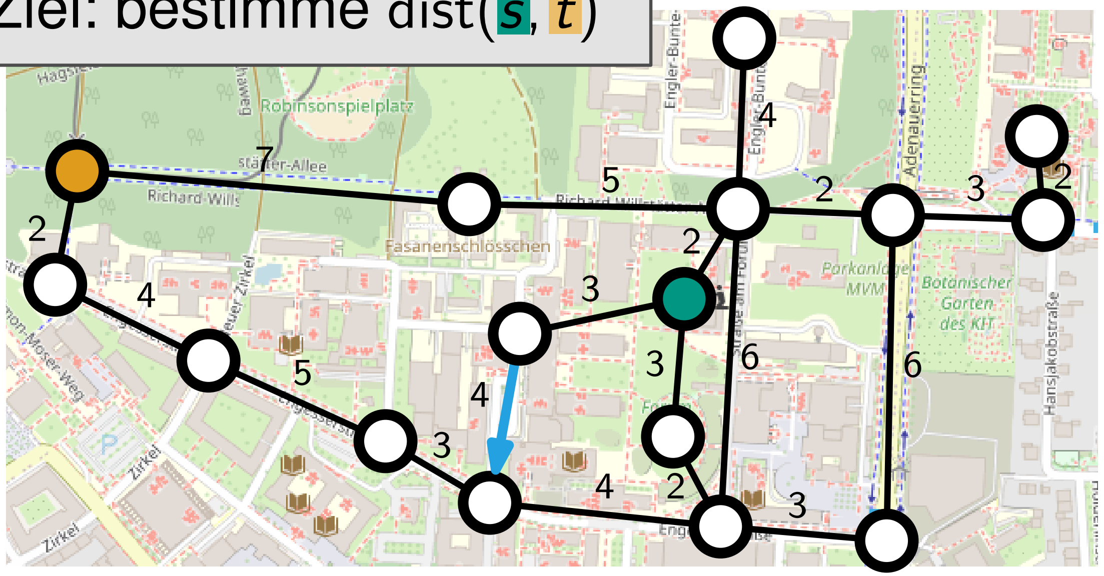
- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?

Neues Setting

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

Dijkstra (zielgerichtet)

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



Neues Setting

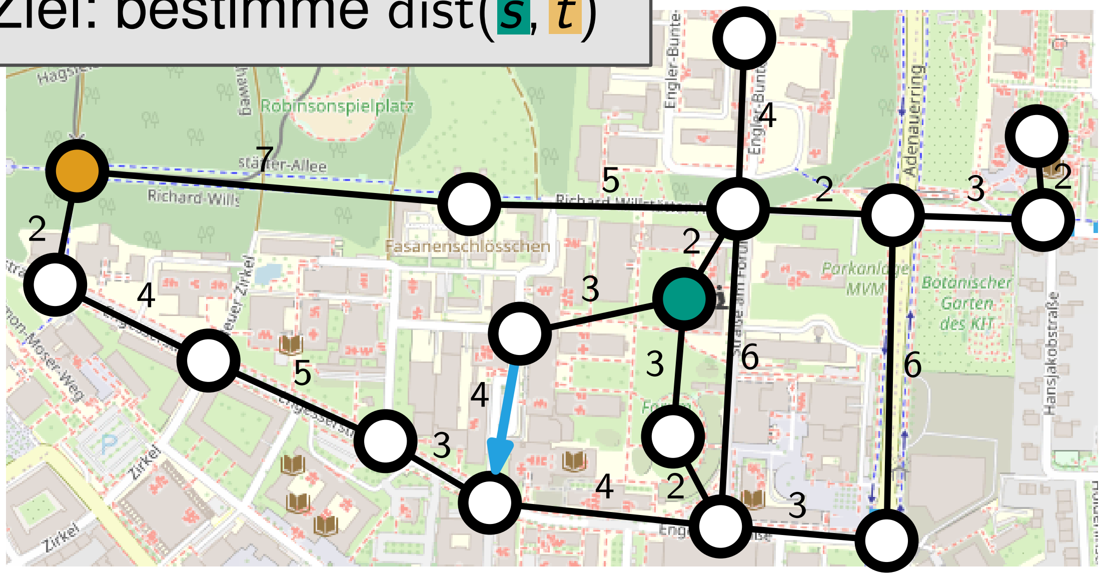
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \hat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

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

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



Neues Setting

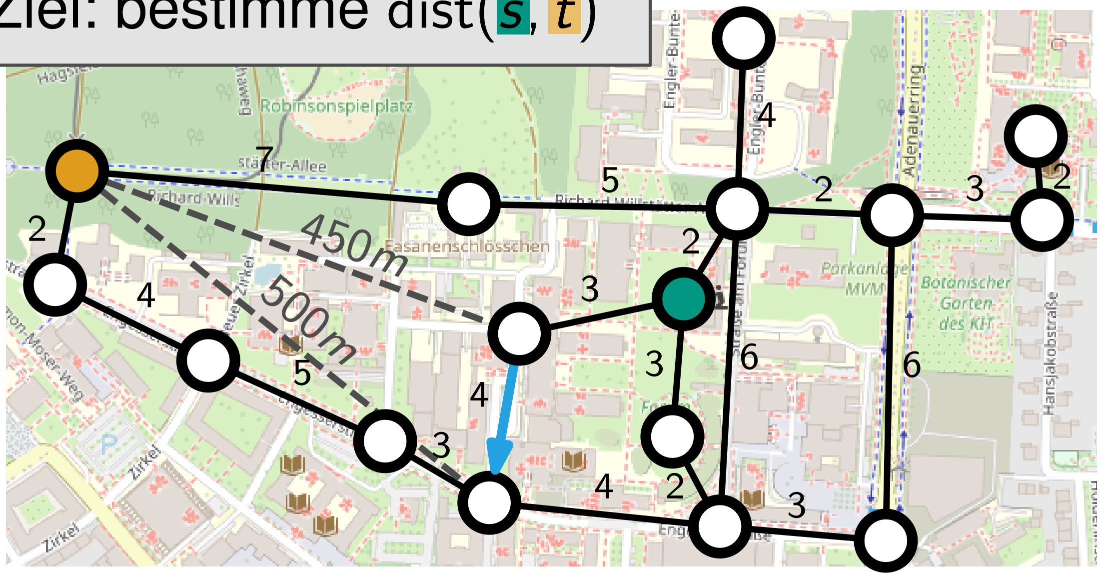
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?

Dijkstra (zielgerichtet)

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



Neues Setting

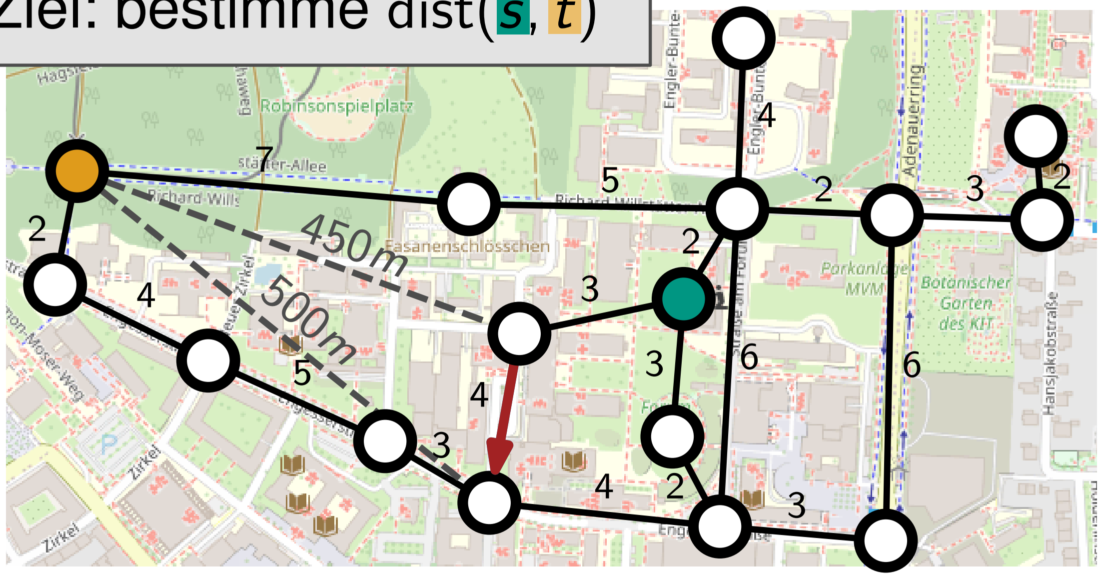
- Straßengraph mit Straßenlänge als Gewichte (Gewicht $1 \widehat{=} 50m$)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?

Dijkstra (zielgerichtet)

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



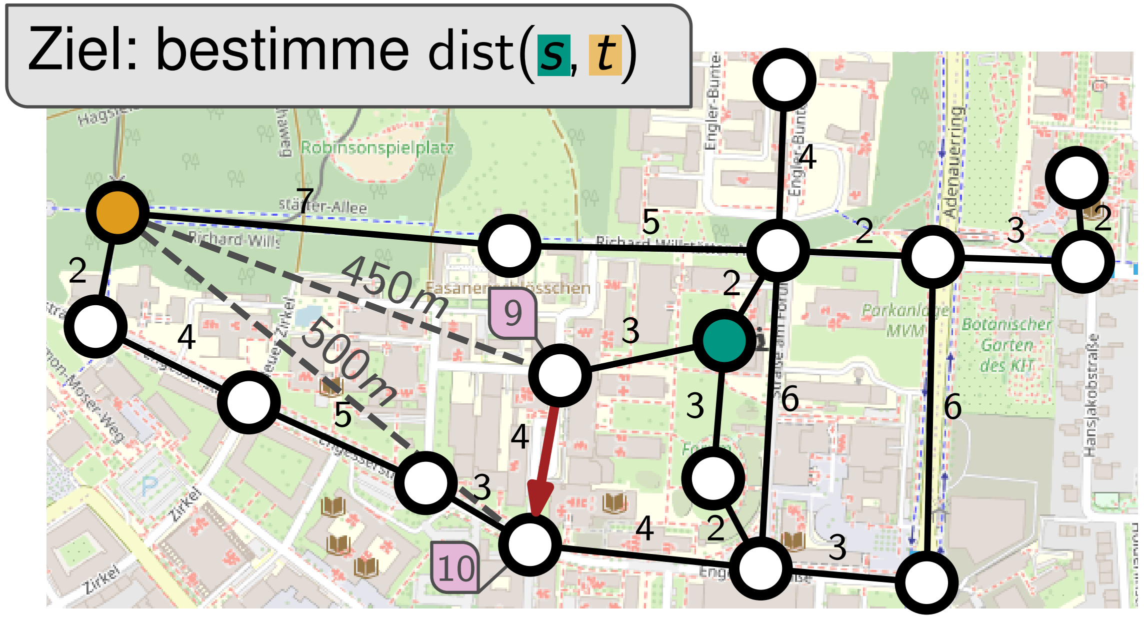
Idee

- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?

Neues Setting

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

Dijkstra (zielgerichtet)



Neues Setting

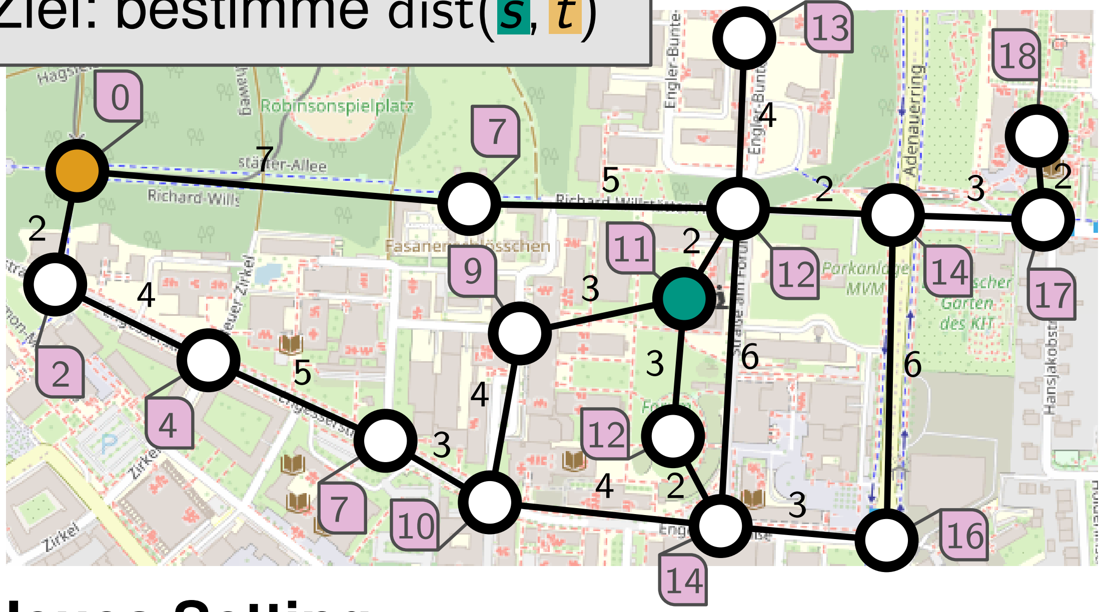
- Straßengraph mit Straßenlänge als Gewichte (Gewicht 1$\hat{=}$50m)
- mehr Infos über den Graphen (z.B. geographische Koordinaten)

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)

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)

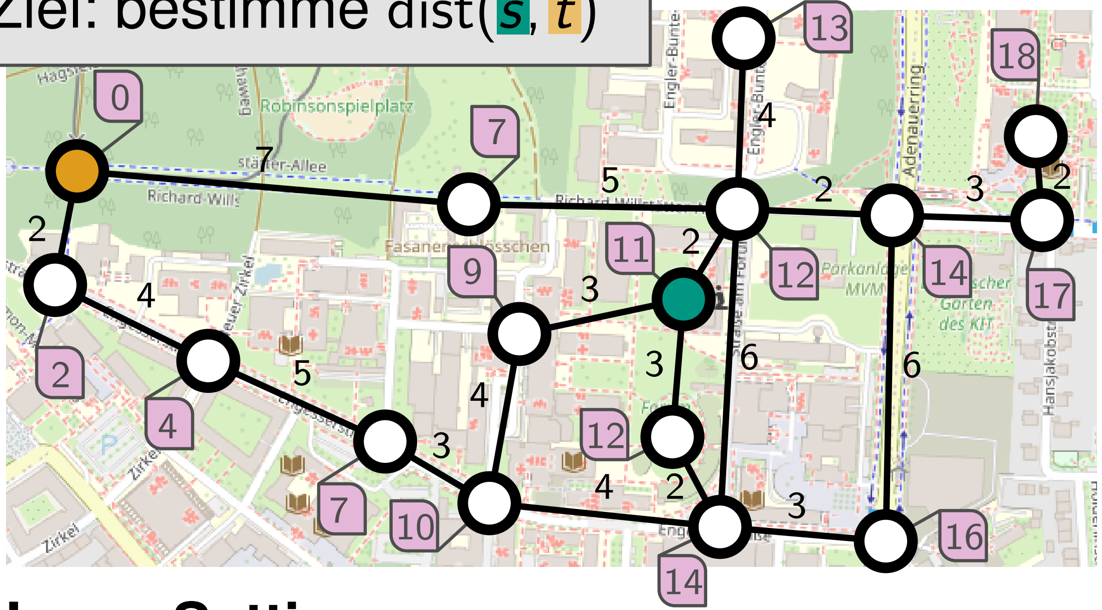
Idee

- 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)

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

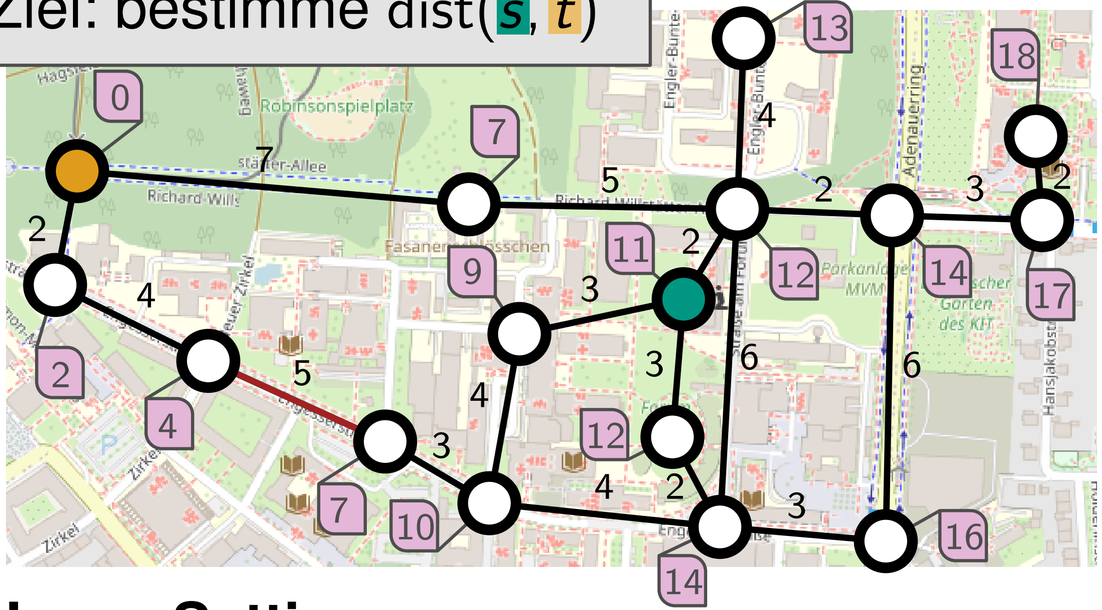
- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

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

Dijkstra (zielgerichtet)

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

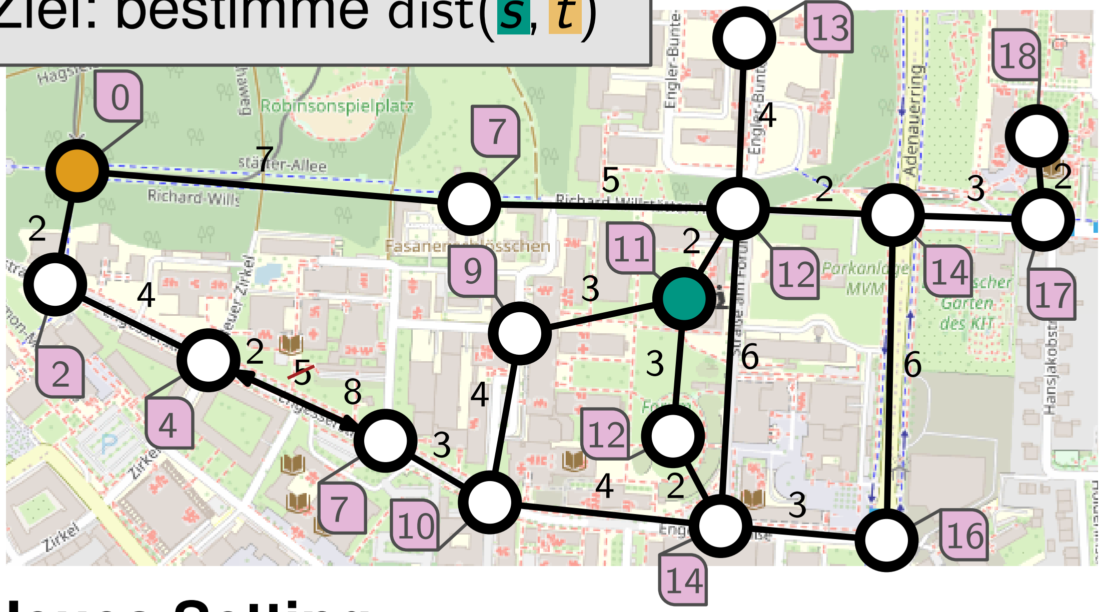
- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

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

Dijkstra (zielgerichtet)

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

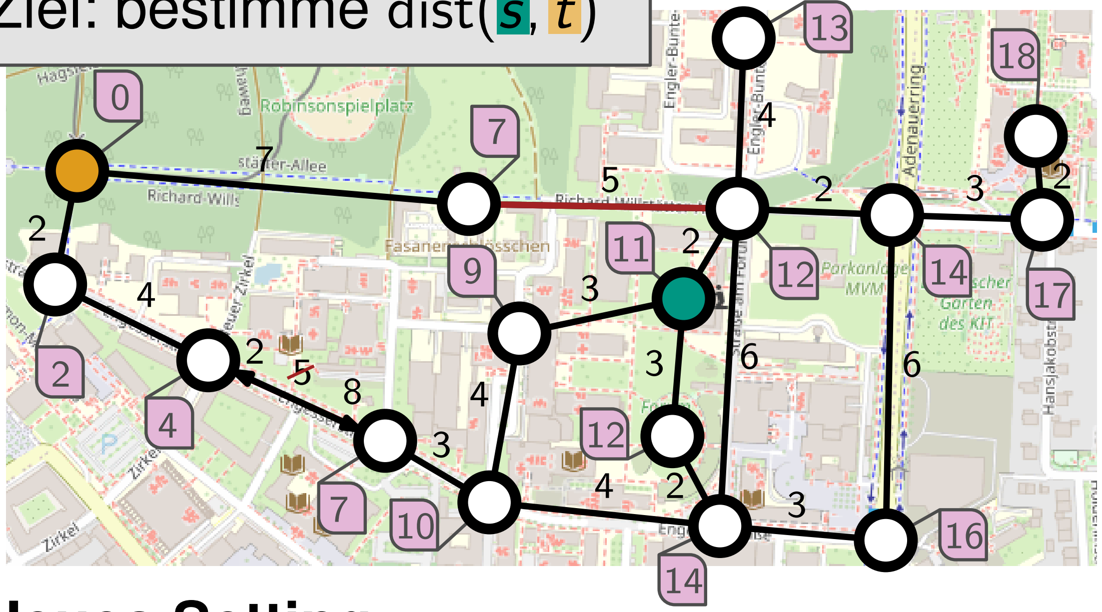
- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

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

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

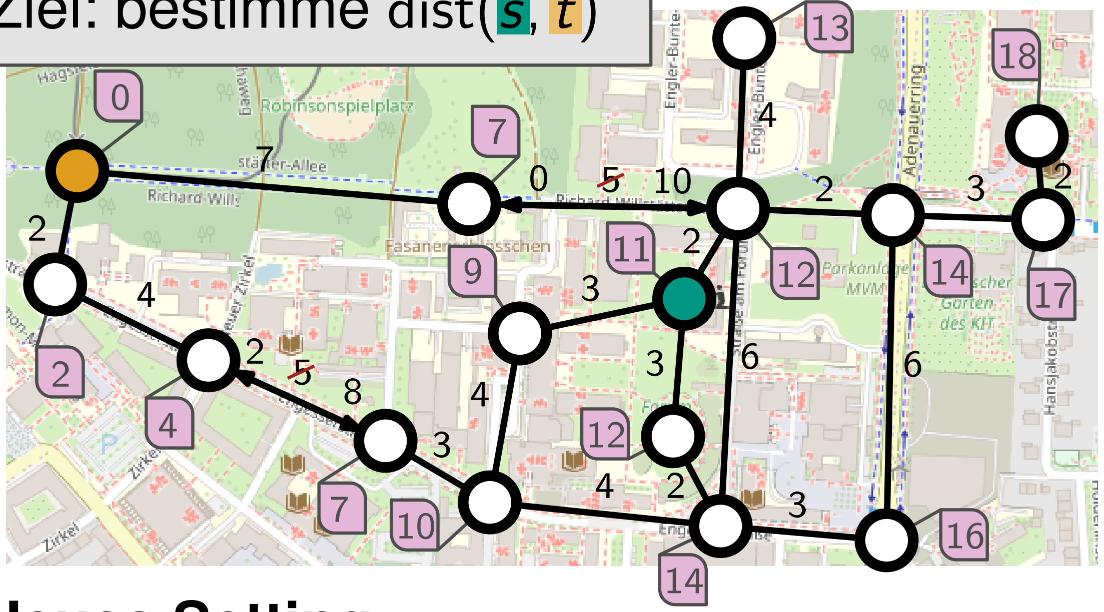
- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

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

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

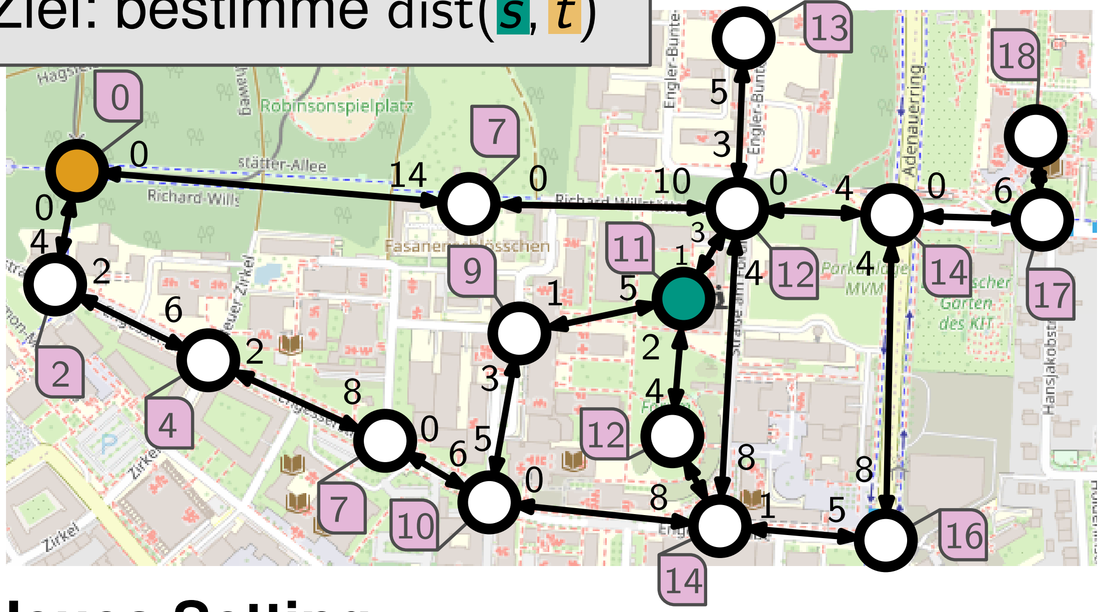
- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

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

Dijkstra (zielgerichtet)

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



Neues Setting

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

Idee

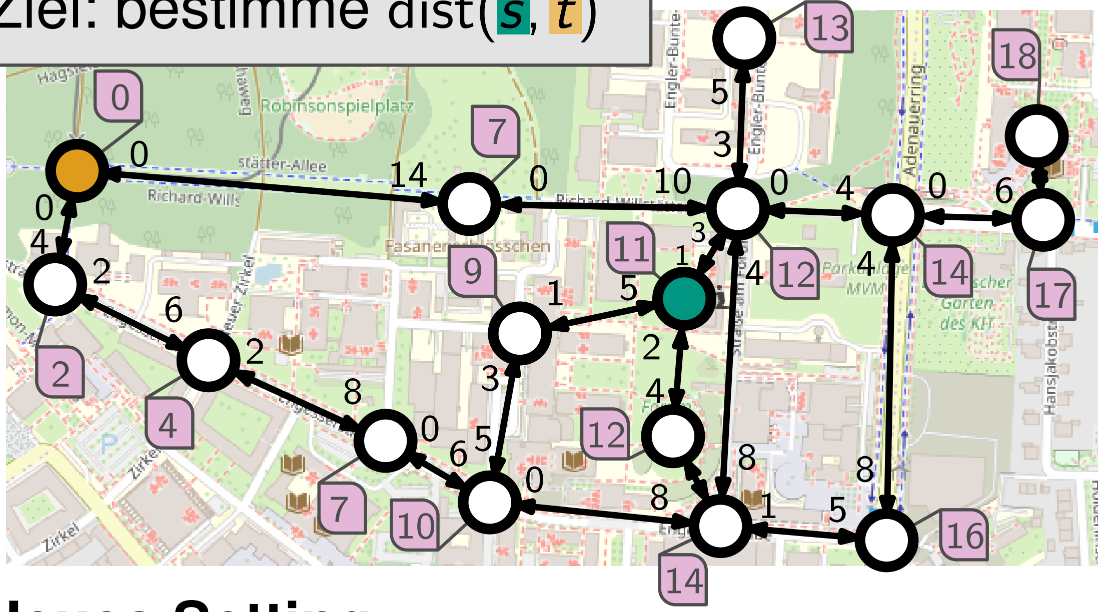
- mache gute Kanten günstiger und schlechte Kanten teurer
- Was sind gute/schlechte Kanten?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

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

Dijkstra (zielgerichtet)

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?
- Wie können wir Kanten bewerten?
- abhängig von euklidischer Distanz $\pi(v)$ zum Ziel

Neue Kantengewichtsfunktion in G^*

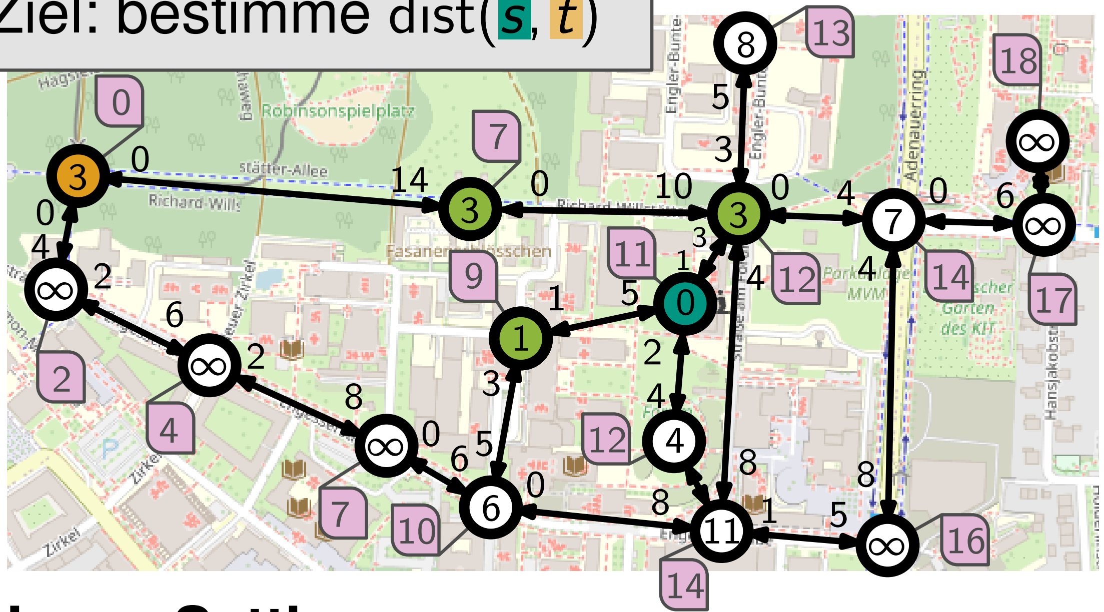
$$\mathrm{len}^*(u, v) = \mathrm{len}(u, v) - \pi(u) + \pi(v)$$

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

Bleibt der kürzeste s-t-Pfad gleich?

Dijkstra (zielgerichtet)

Ziel: bestimme dist(s, t)



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

Dijkstra (zielgerichtet): Korrektheit

- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G

Neue Kantengewichtsfunktion in G^*

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

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?

Neue Kantengewichtsfunktion in G^*

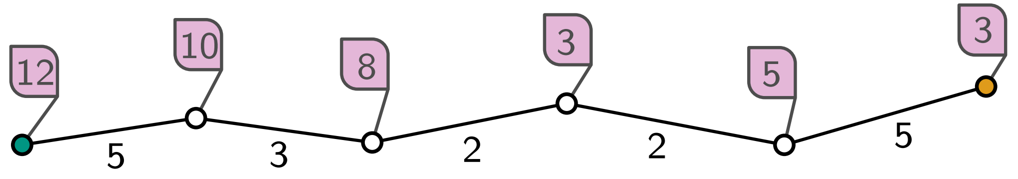
$$\text{len}^*(u, v) = \text{len}(u, v) - \pi(u) + \pi(v)$$

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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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



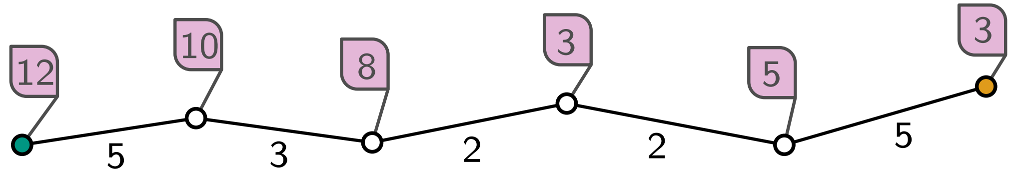
Dijkstra (zielgerichtet): Korrektheit

- 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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

Neue Kantengewichtsfunktion in G^*

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



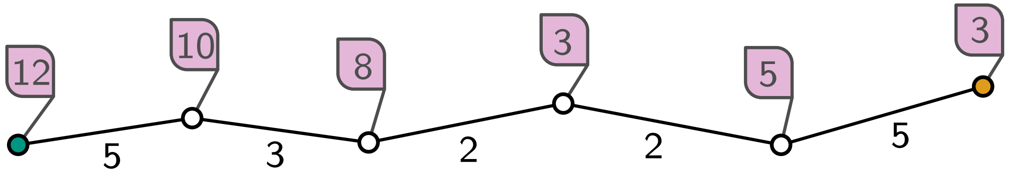
Dijkstra (zielgerichtet): Korrektheit

- 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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

Neue Kantengewichtsfunktion in G^*

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

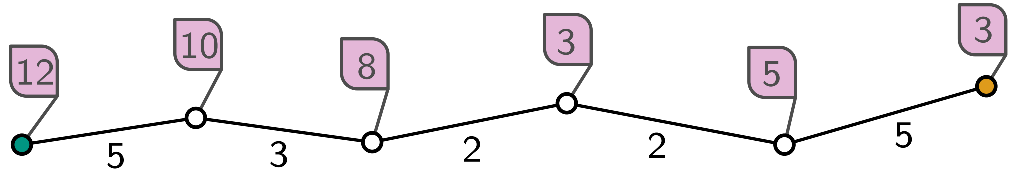


Dijkstra (zielgerichtet): Korrektheit

- 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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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



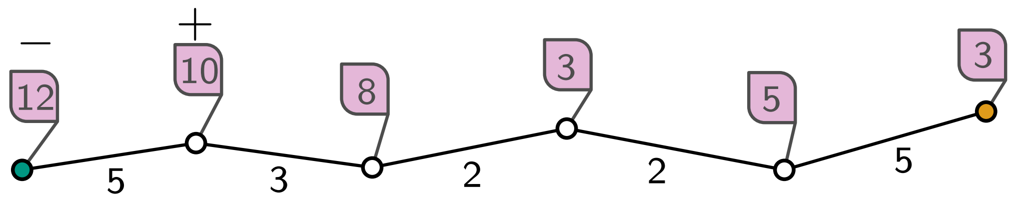
$$\begin{aligned}
\text{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \text{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\text{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \text{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1}))
\end{aligned}$$

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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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



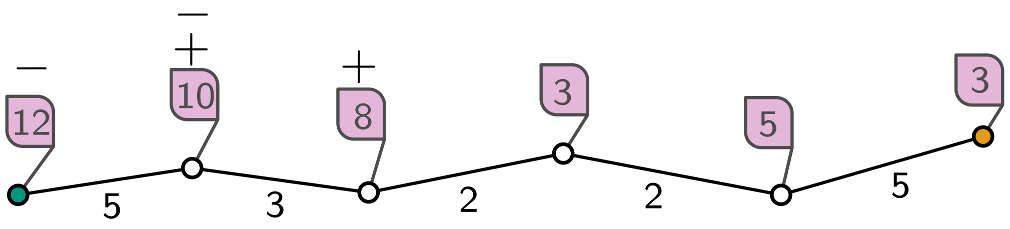
$$\begin{aligned}
\text{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \text{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\text{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \text{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1}))
\end{aligned}$$

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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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



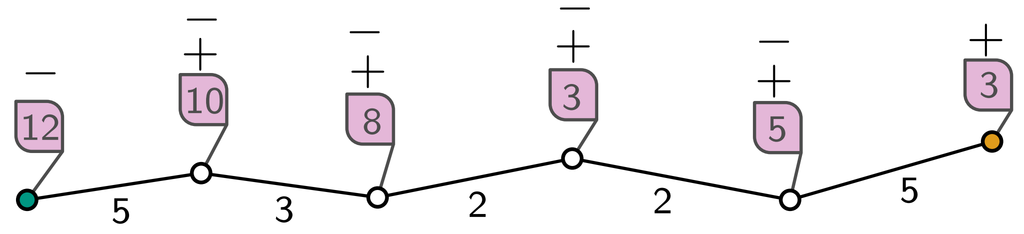
$$\begin{aligned}\text{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \text{len}^*(v_i, v_{i+1}) \\ &= \textstyle\sum_{i=1}^{k-1} (\text{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\ &= \textstyle\sum_{i=1}^{k-1} \text{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1}))\end{aligned}$$

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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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



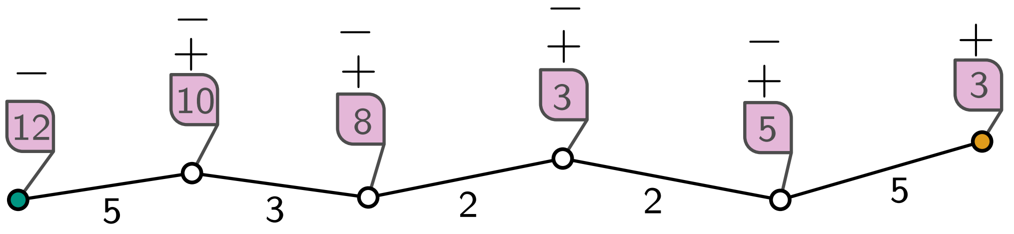
$$\begin{aligned}
\mathrm{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \mathrm{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\mathrm{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \mathrm{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1}))
\end{aligned}$$

Dijkstra (zielgerichtet): Korrektheit

- 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^*

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



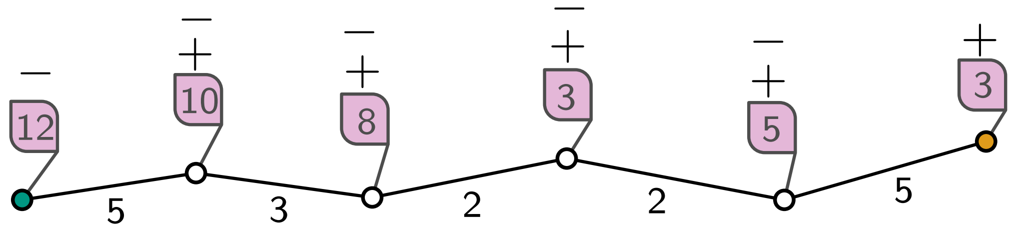
$$\begin{aligned}
\mathrm{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \mathrm{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\mathrm{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \mathrm{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1})) \\
&= \mathrm{len}(P)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

Dijkstra (zielgerichtet): Korrektheit

- 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^*

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



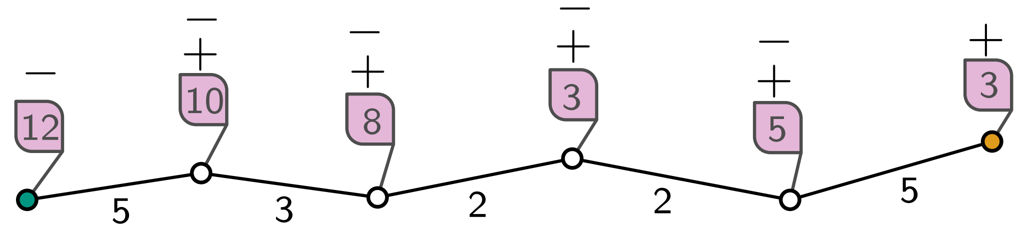
$$
\begin{aligned}
\mathrm{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \mathrm{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\mathrm{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \mathrm{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1})) \\
&= \mathrm{len}(P)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

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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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



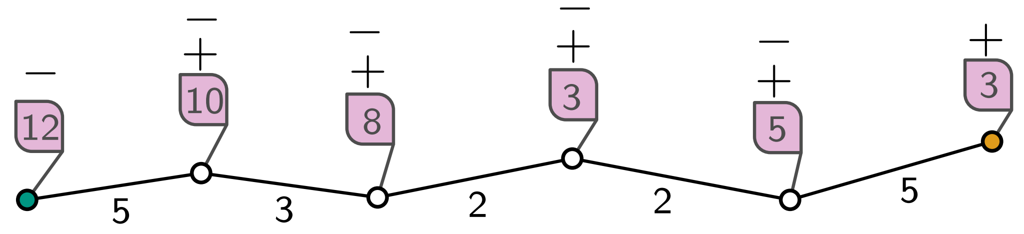
$$
\begin{aligned}
\text{len}^*(P) &= \textstyle\sum_{i=1}^{k-1} \text{len}^*(v_i, v_{i+1}) \\
&= \textstyle\sum_{i=1}^{k-1} (\text{len}(v_i, v_{i+1}) - \pi(v_i) + \pi(v_{i+1})) \\
&= \textstyle\sum_{i=1}^{k-1} \text{len}(v_i, v_{i+1}) + \sum_{i=1}^{k-1} (-\pi(v_i) + \pi(v_{i+1})) \\
&= \text{len}(P)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

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 $P = (s = v_1, v_2, v_3, \dots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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



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

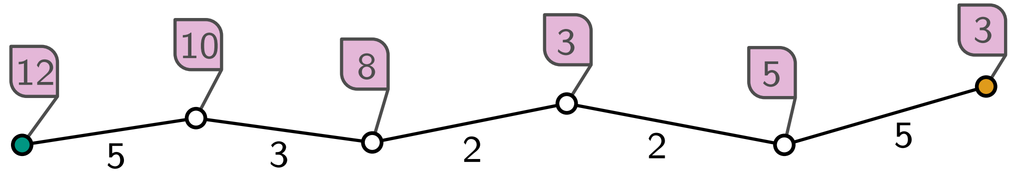
Dijkstra (zielgerichtet): Korrektheit

- 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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

Neue Kantengewichtsfunktion in G^*

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

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



Dijkstra (zielgerichtet): Korrektheit

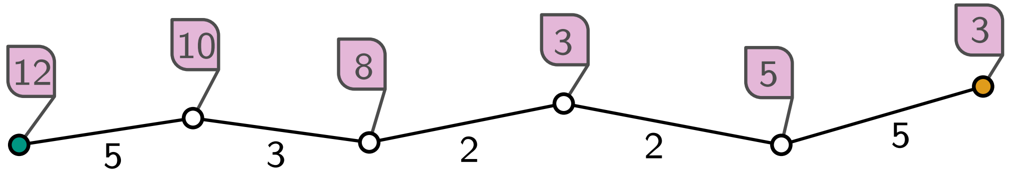
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \dots, v_k = t)$

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

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

Neue Kantengewichtsfunktion in G^*

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



Dijkstra (zielgerichtet): Korrektheit

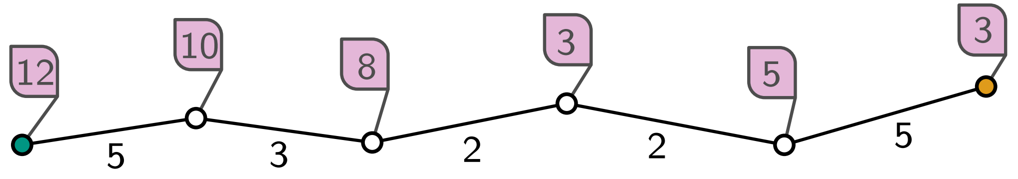
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?

Neue Kantengewichtsfunktion in G^*

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

- 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 $\pi : V \to \mathbb{R}$

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 $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

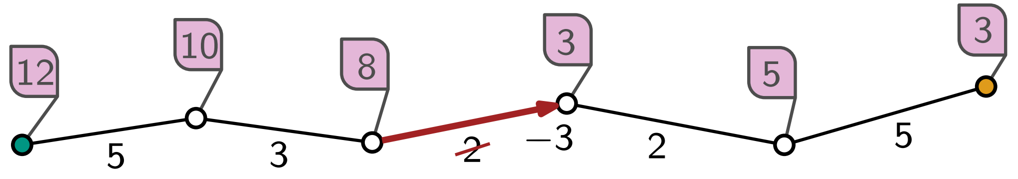
$$\mathrm{len}^*(P) = \mathrm{len}(P) \underbrace{- \pi(s) + \pi(t)}_{\text{konstant für alle } s\text{-}t\text{-Pfade}}$$

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

Neue Kantengewichtsfunktion in G^*

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

↯ negative Kantengewichte



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

Dijkstra (zielgerichtet): Korrektheit

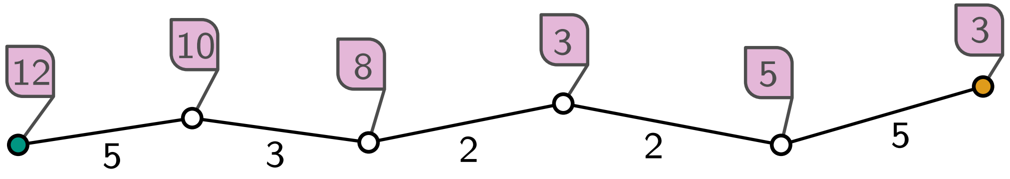
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

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

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?

Dijkstra (zielgerichtet): Korrektheit

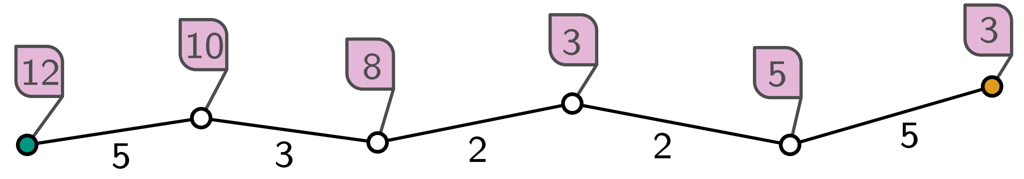
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

- 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 $(u, v) \in E$

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?

Dijkstra (zielgerichtet): Korrektheit

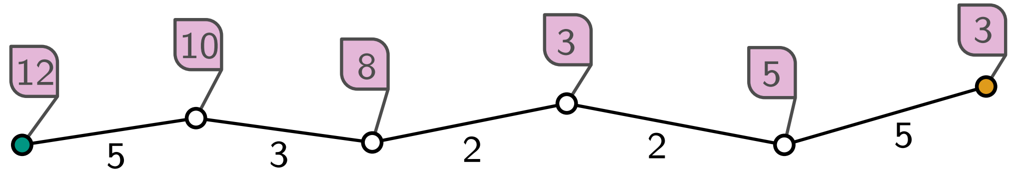
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

- 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?

Dijkstra (zielgerichtet): Korrektheit

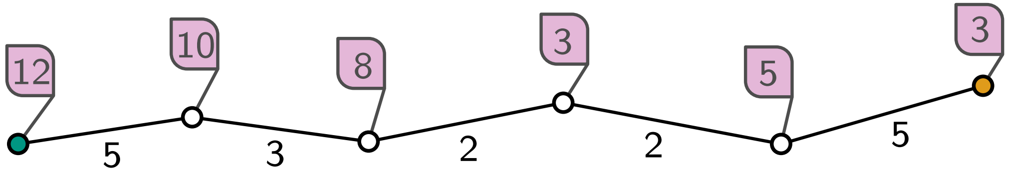
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \dots, v_k = t)$

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

- 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 $\text{dist}(u, t)$

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?

Dijkstra (zielgerichtet): Korrektheit

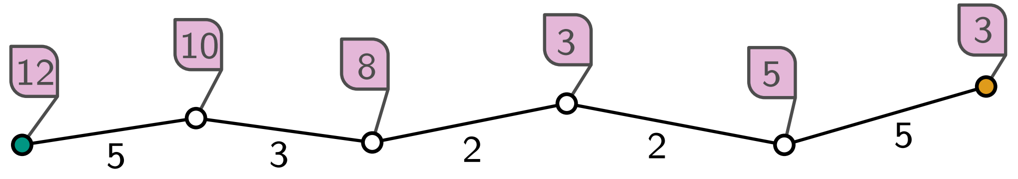
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

- 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

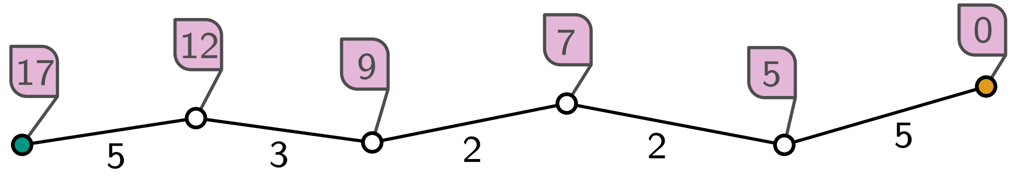
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?

Neue Kantengewichtsfunktion in G^*

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

- 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 $\pi : V \to \mathbb{R}$

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

- 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 $\text{dist}(u, t)$

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

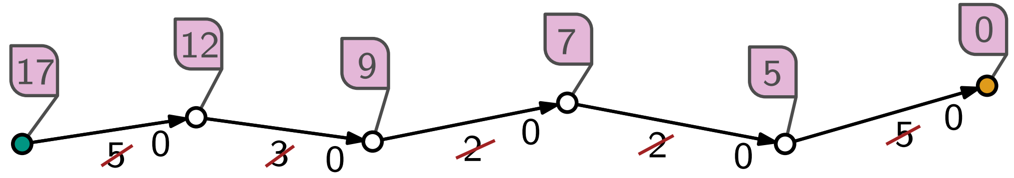
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

- 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

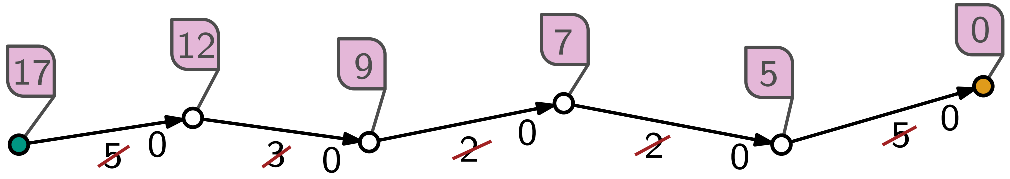
- zu zeigen: kürzester s-t-Pfad in G^* ist auch kürzester s-t-Pfad in G
- Wie verändert sich die Länge eines s-t-Pfades?
- s-t-Pfad $P = (s = v_1, v_2, v_3, \ldots, v_k = t)$

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

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

Neue Kantengewichtsfunktion in G^*

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



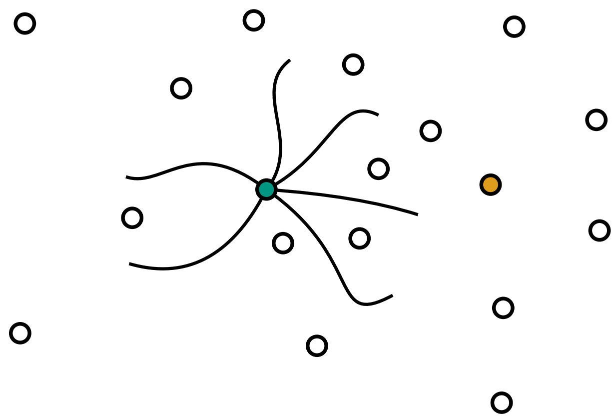
Findet Dijkstra mit jedem π den kürzesten s-t-Weg?

Ist euklidische Distanz auf Straßengraphen eine gute Potentialfunktion?

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

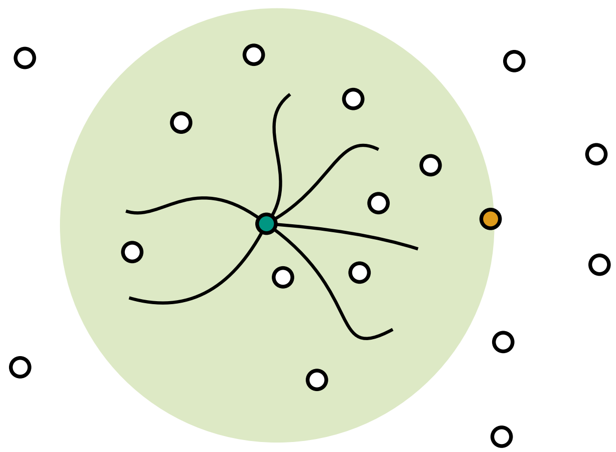
Dijkstra (zielgerichtet)

- Dijkstra sucht viel ab, um Ziel zu finden



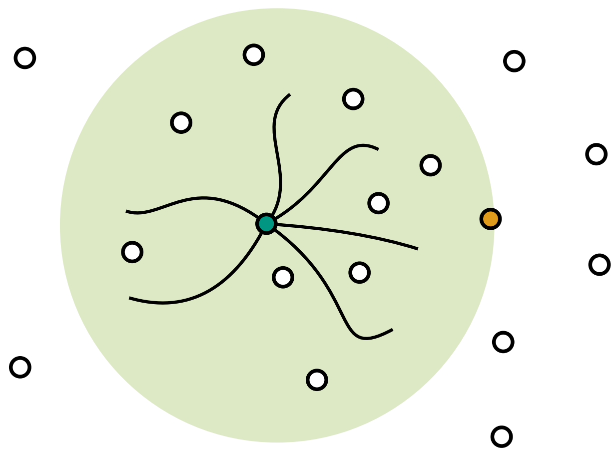
Dijkstra (zielgerichtet)

- Dijkstra sucht viel ab, um Ziel zu finden



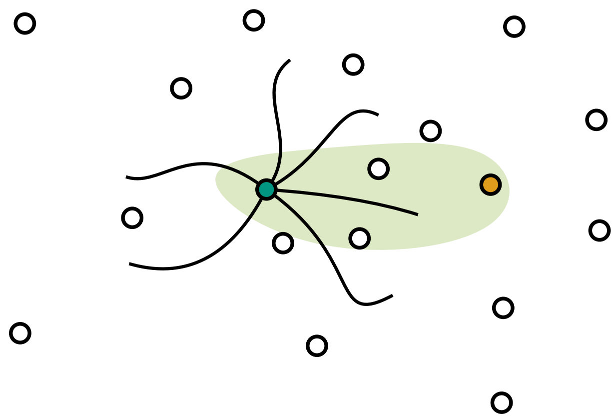
Dijkstra (zielgerichtet)

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



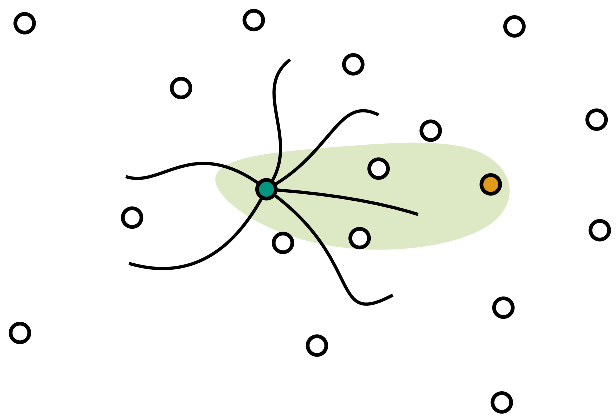
Dijkstra (zielgerichtet)

- 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

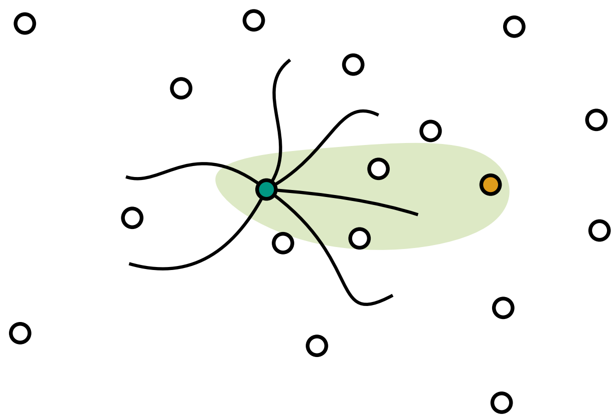


Anwendungen

- verallgemeinerte Variante: A*

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

Evaluation

Karlsruher Institut für Technologie

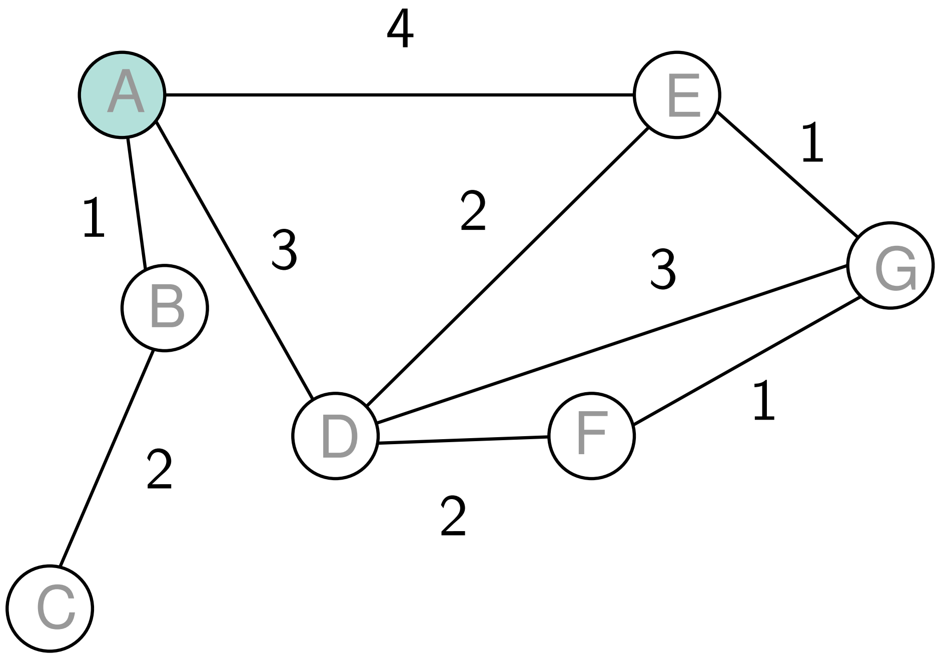
Beschränkte Kantengewichte

Karlsruher Institut für Technologie

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s

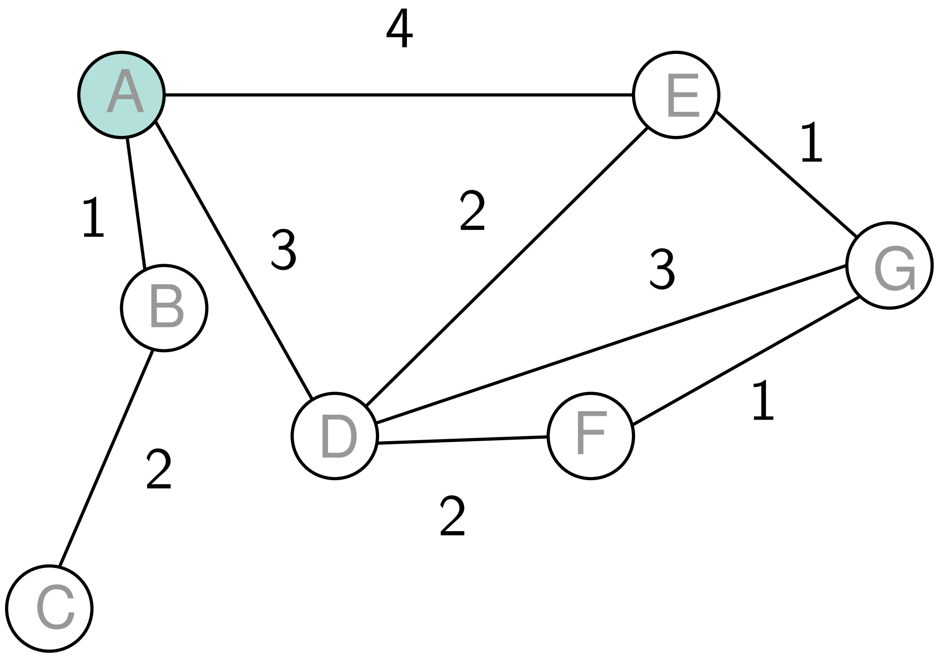
Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s

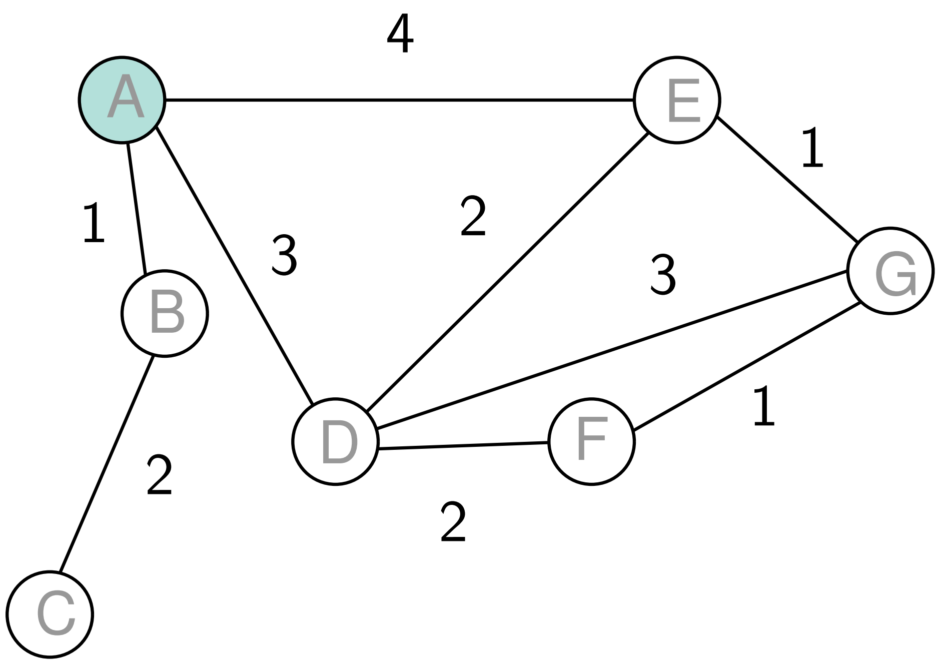


Lösung

Beschränkte Kantengewichte

Karlsruher Institut für Technologie

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s

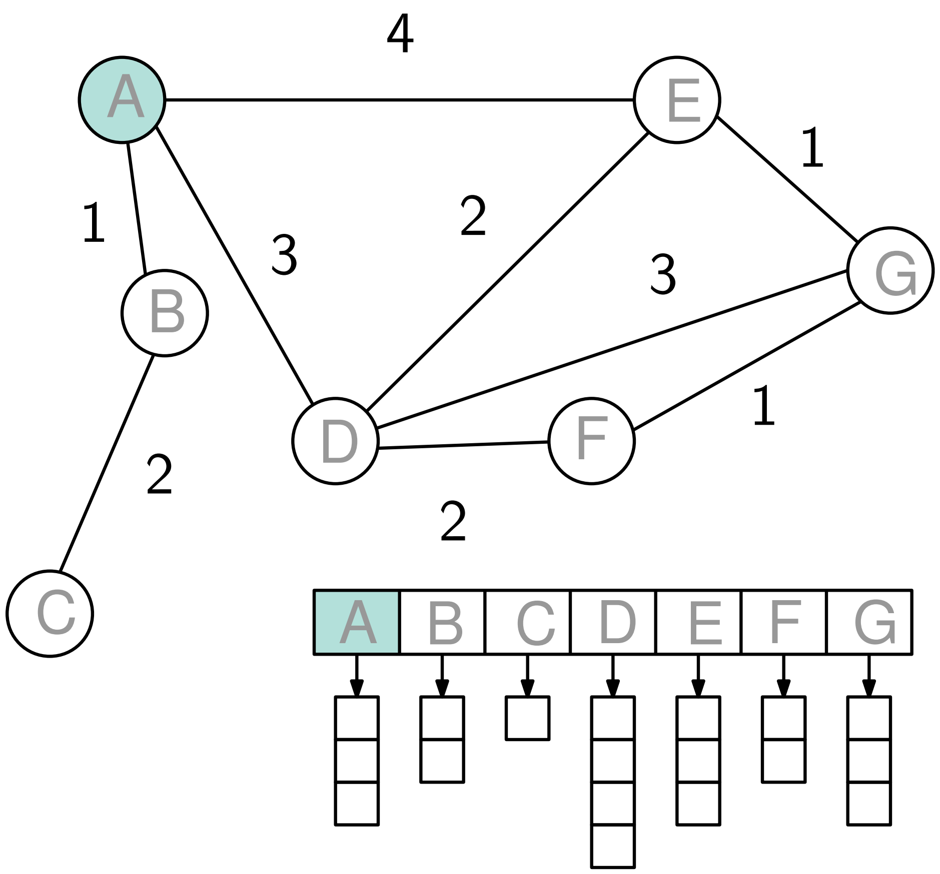


Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1

Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



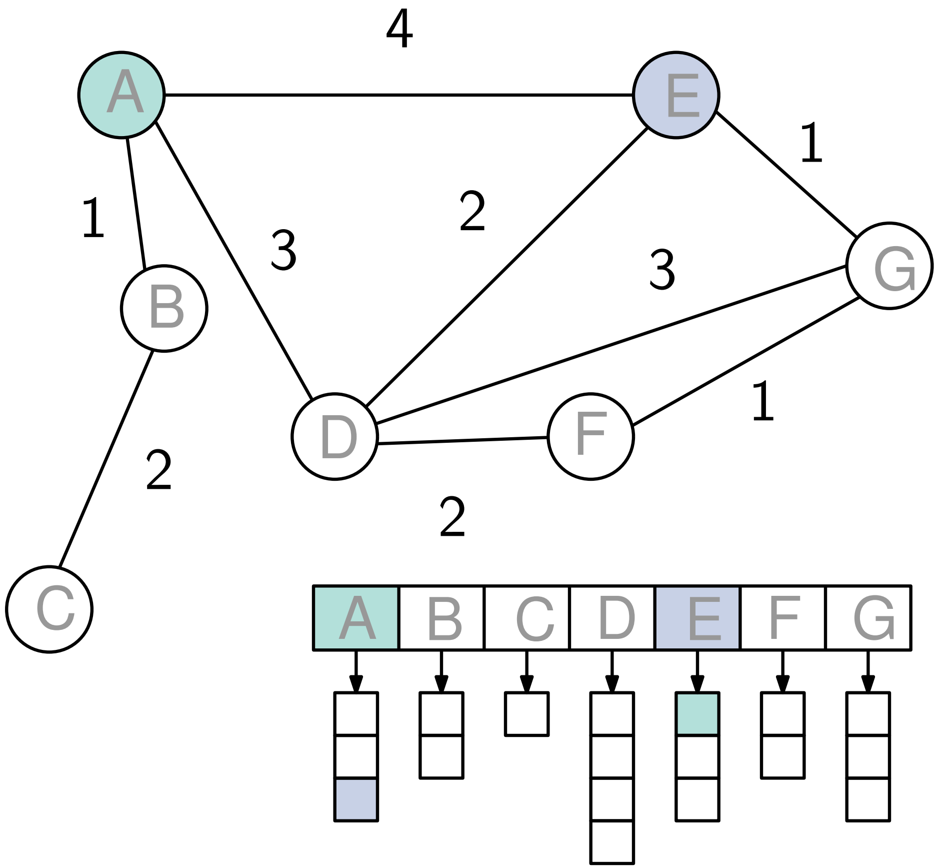
Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1

Beschränkte Kantengewichte

Karlsruher Institut für Technologie

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s

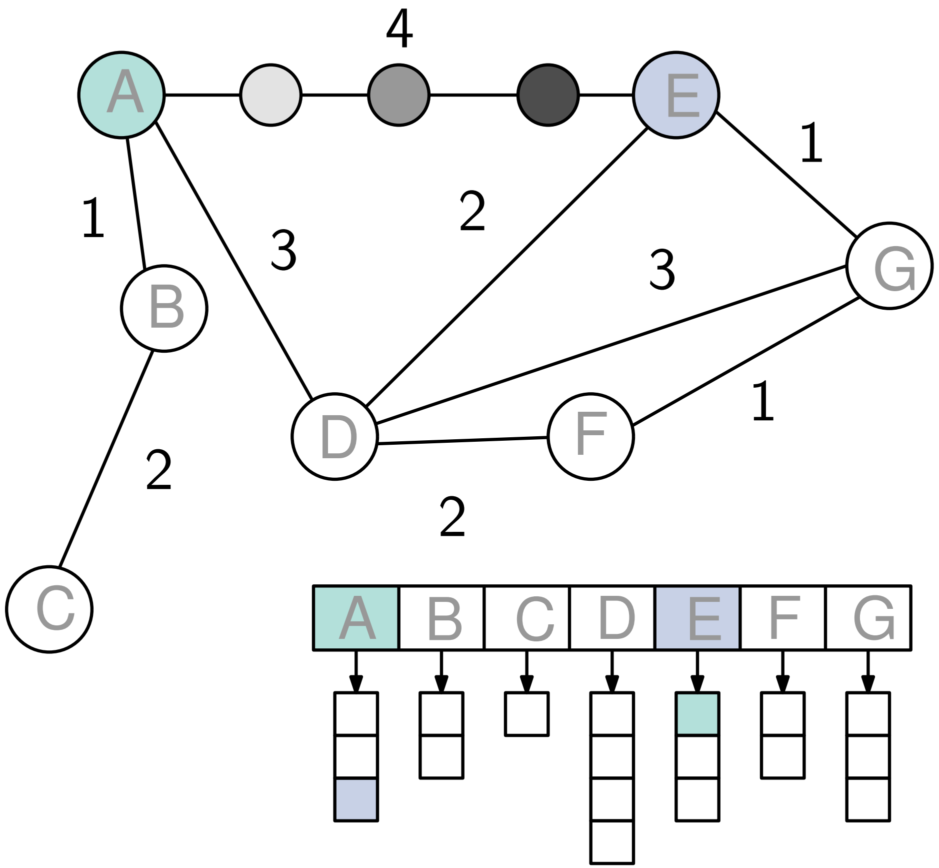


Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1

Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s

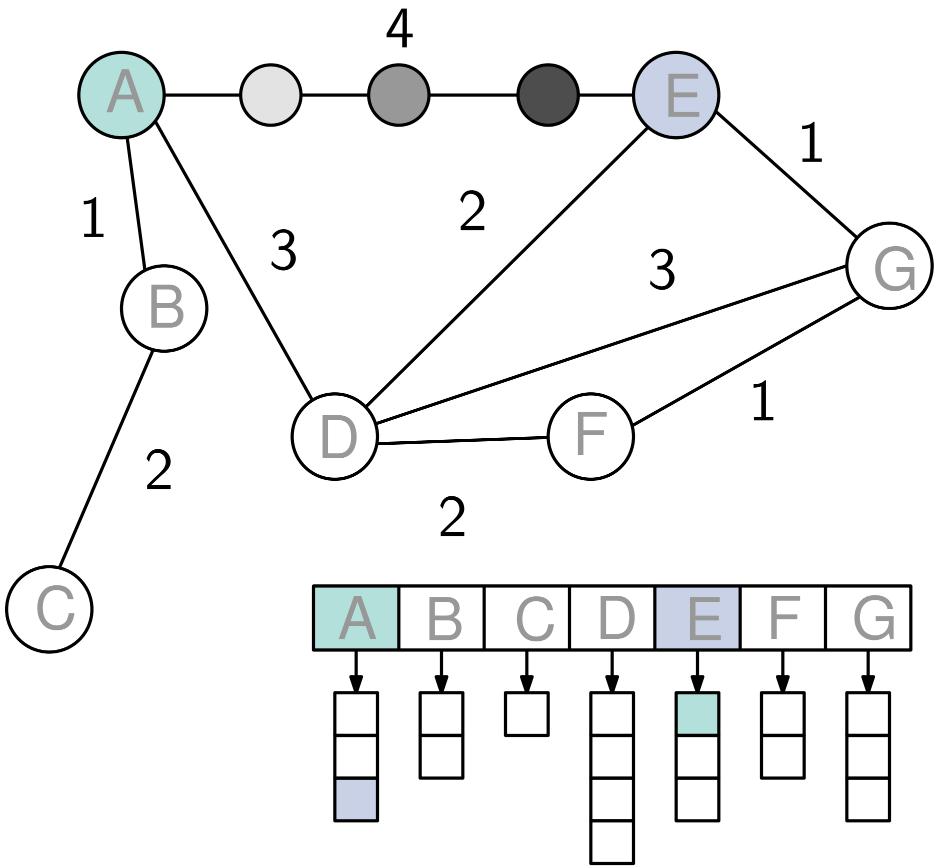


Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1

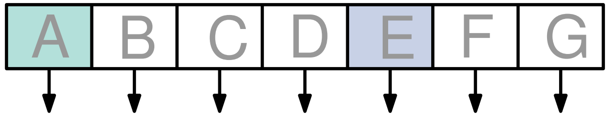
Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



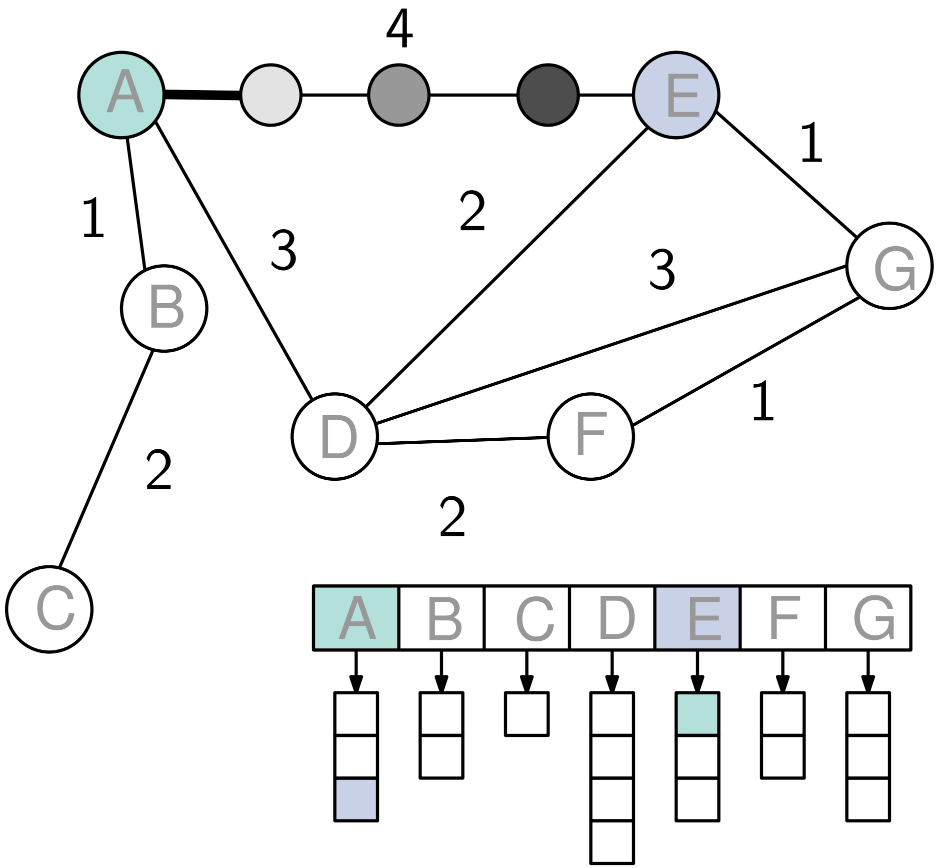
Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1



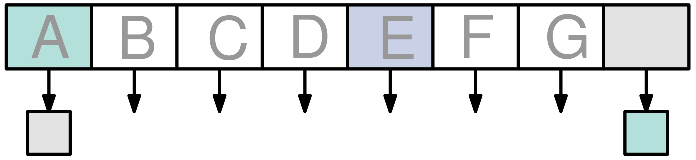
Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



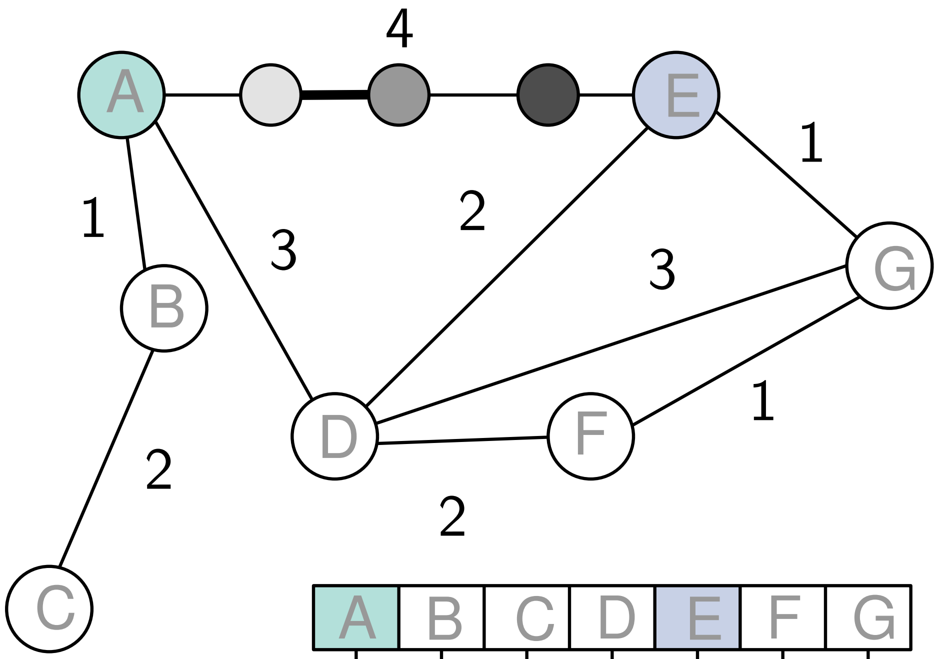
Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1



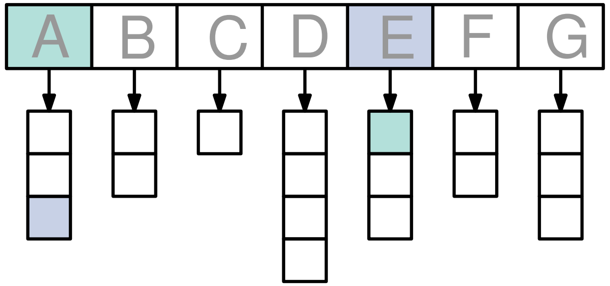
Beschränkte Kantengewichte

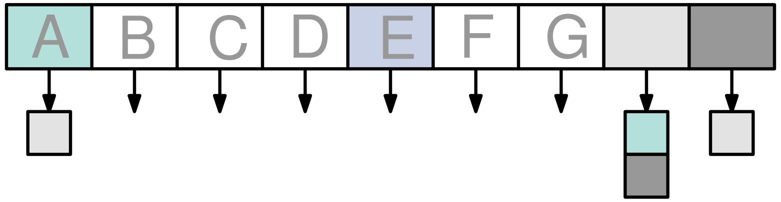
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Lösung

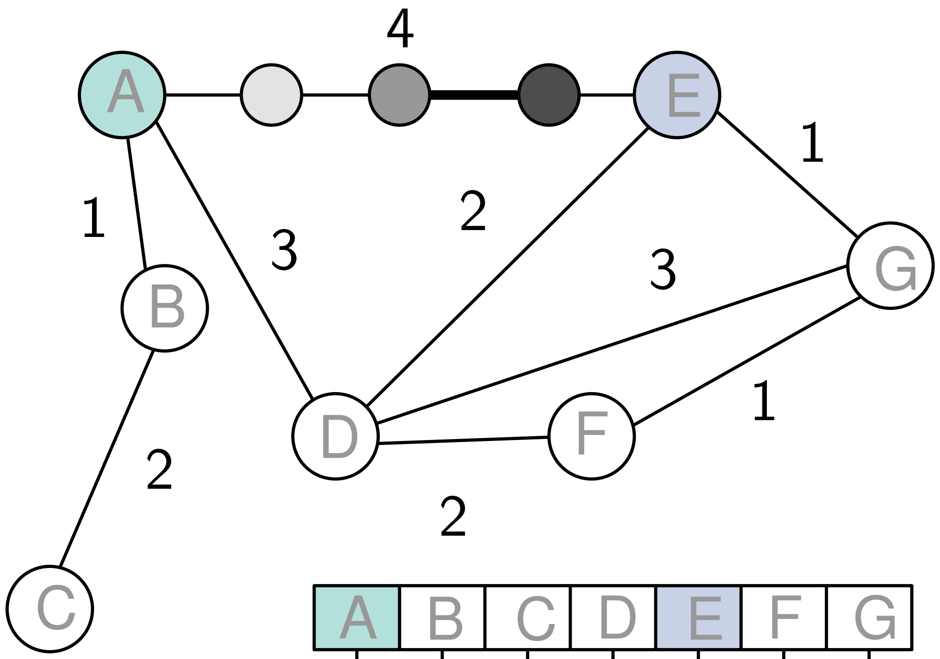
- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1





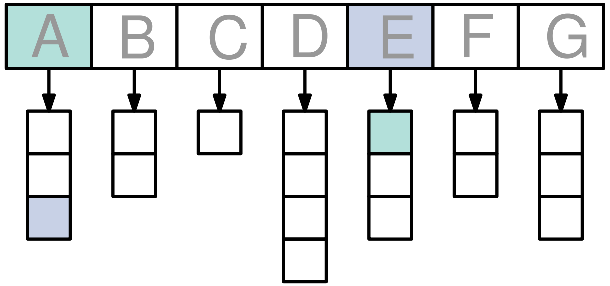
Beschränkte Kantengewichte

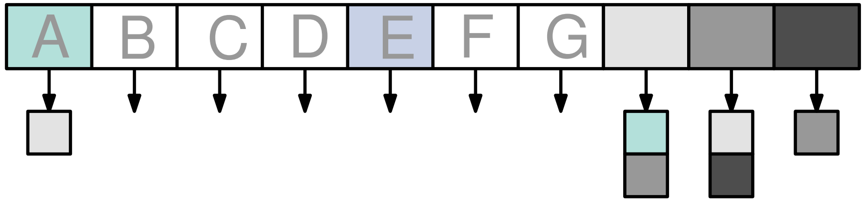
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Lösung

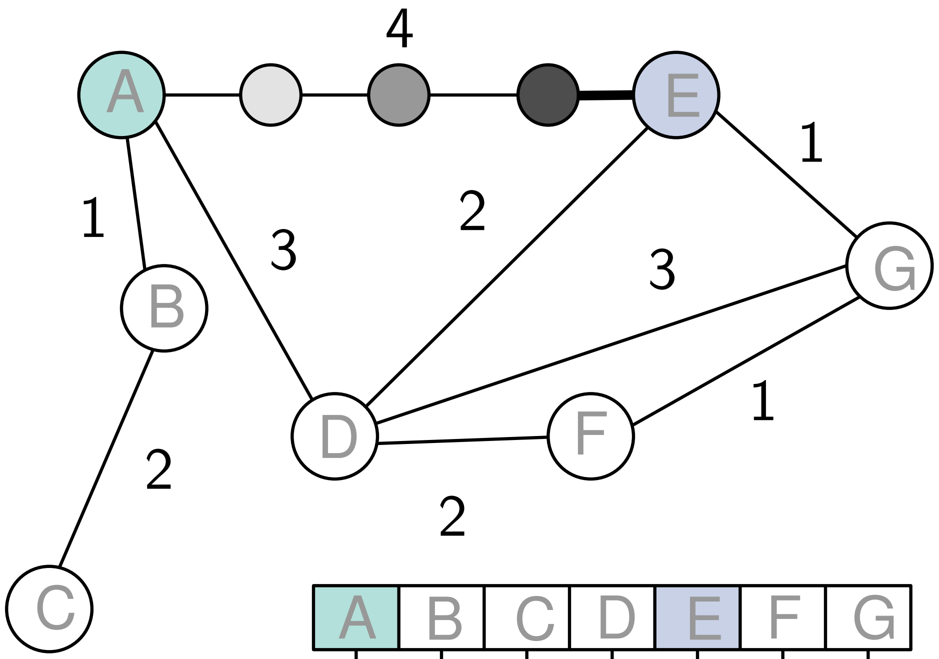
- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1





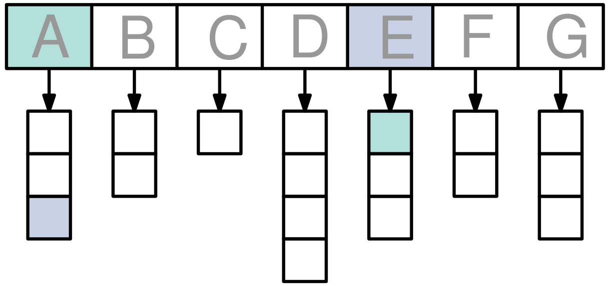
Beschränkte Kantengewichte

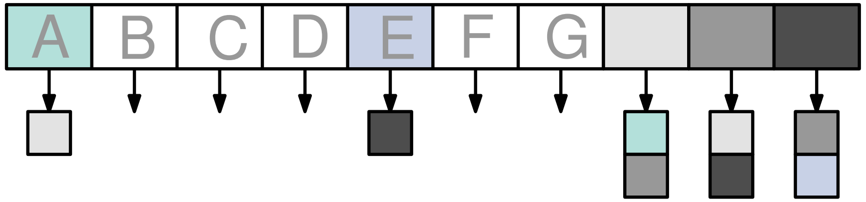
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Lösung

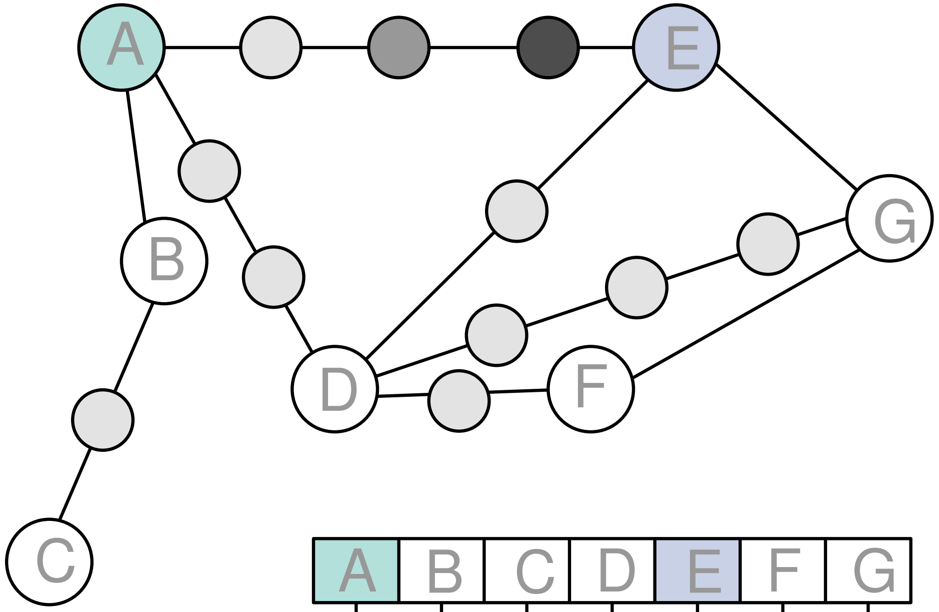
- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1

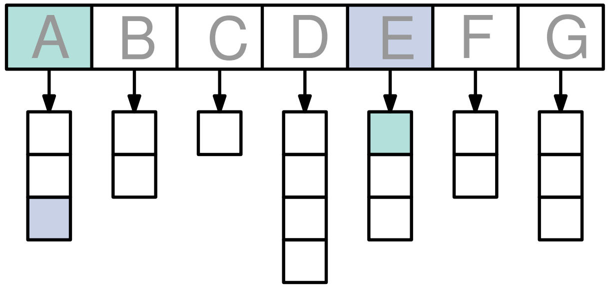




Beschränkte Kantengewichte

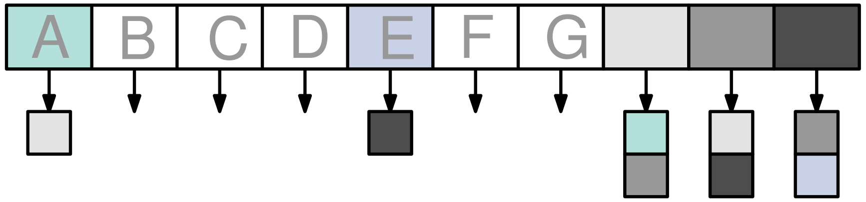
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s





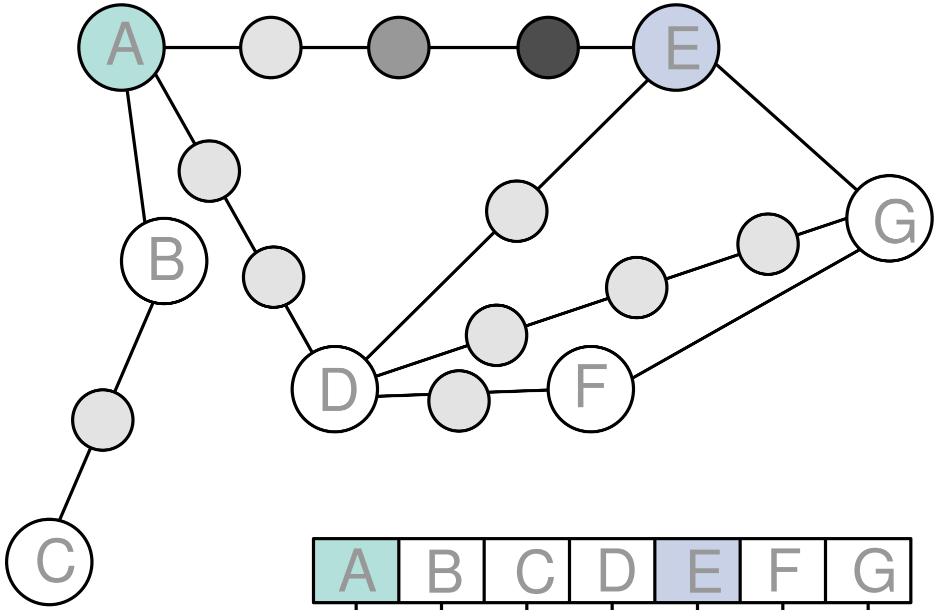
Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1



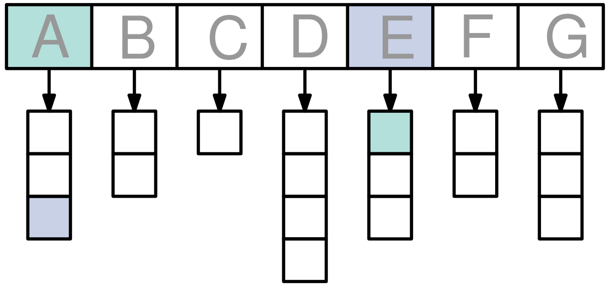
Beschränkte Kantengewichte

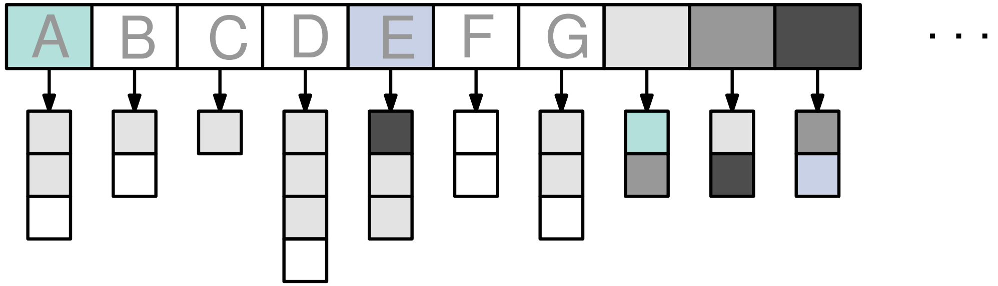
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Lösung

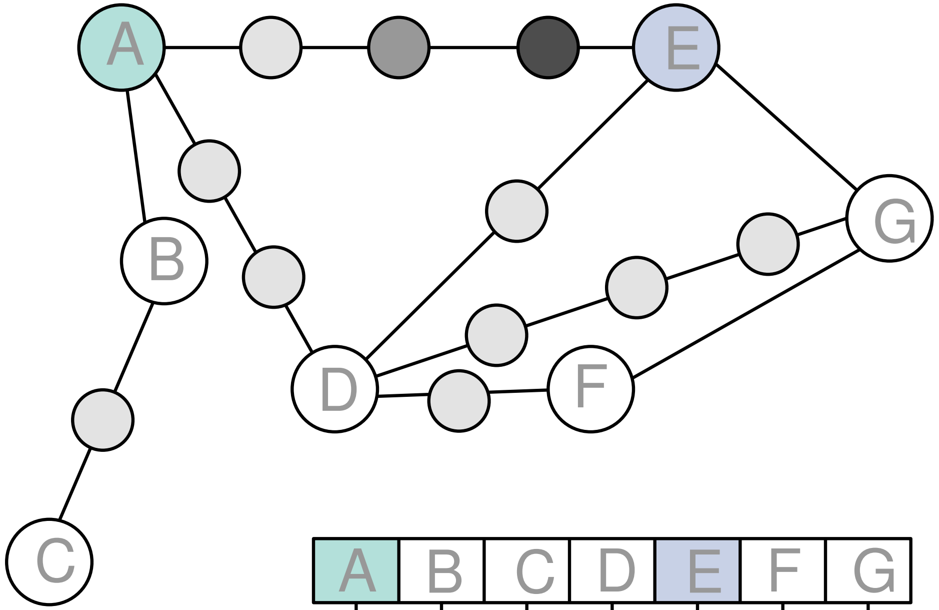
- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1





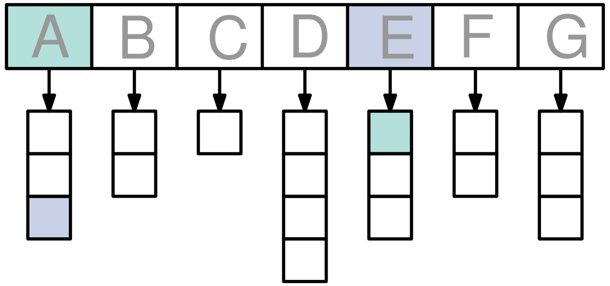
Beschränkte Kantengewichte

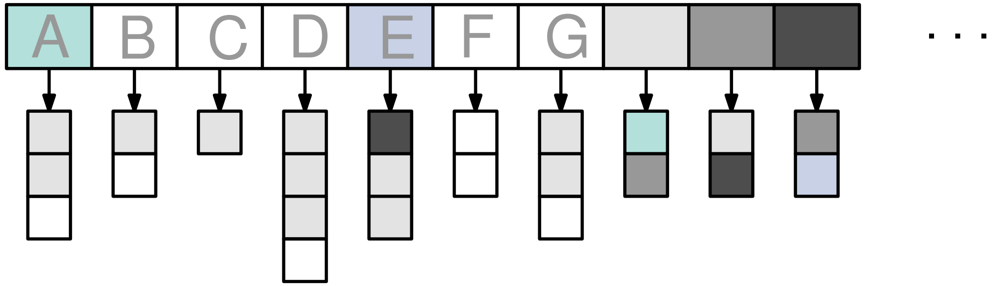
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Lösung

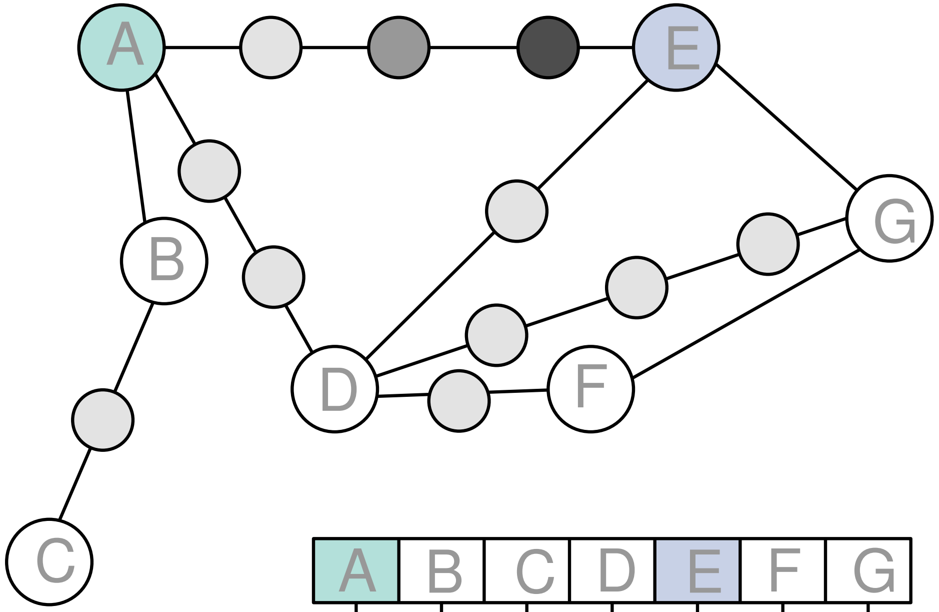
- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1
- BFS auf ungewichtetem Graph von Startknoten s





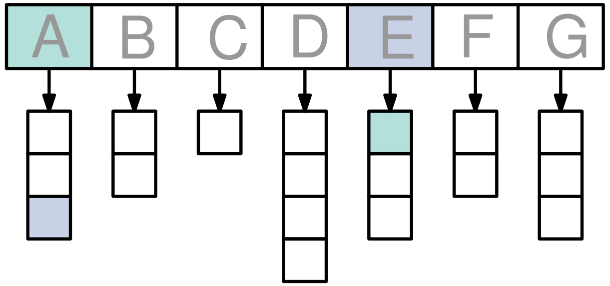
Beschränkte Kantengewichte

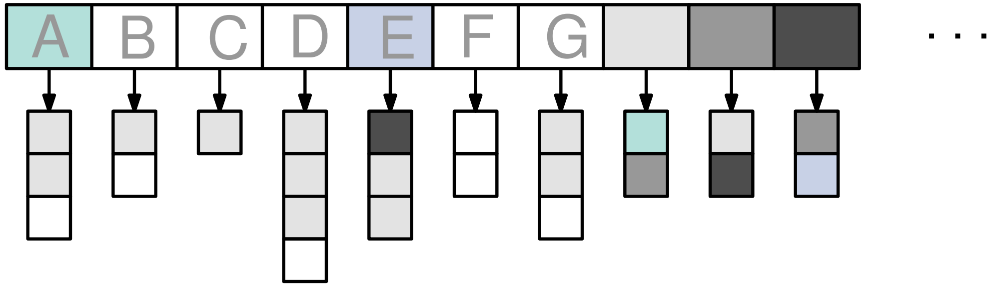
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1
- BFS auf ungewichtetem Graph von Startknoten s
- $O(n + k \cdot m)$ Knoten, $O(k \cdot m)$ Kanten

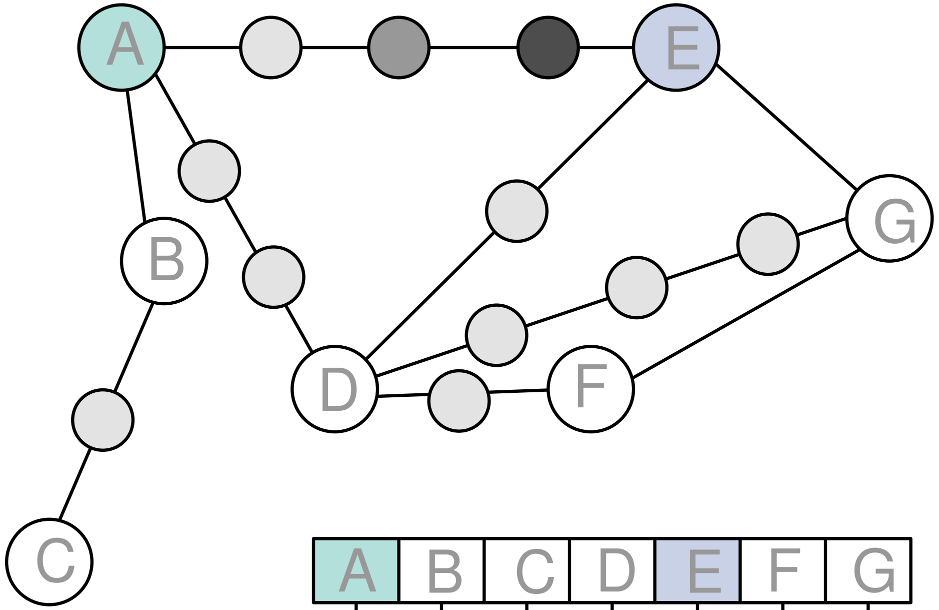




Beschränkte Kantengewichte

Karlsruher Institut für Technologie

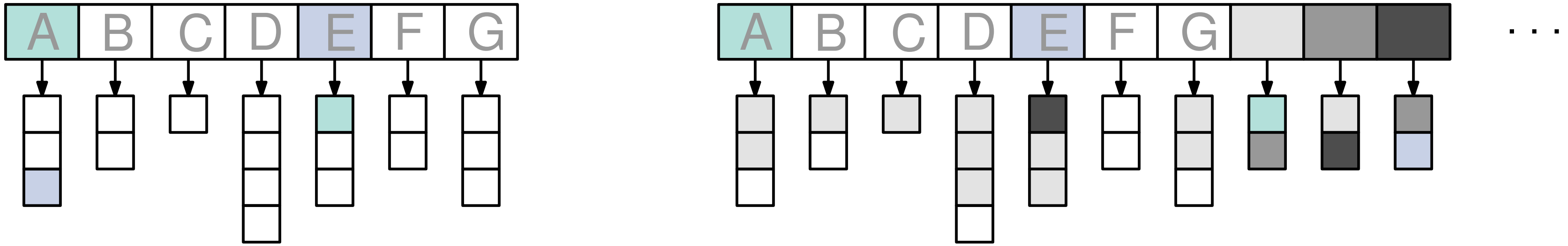
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s



Lösung

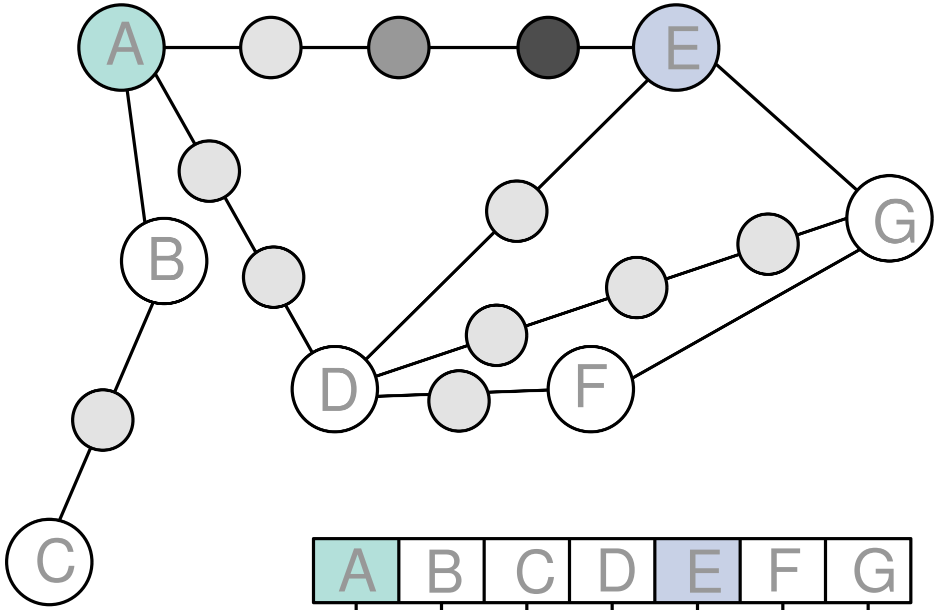
- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1
- BFS auf ungewichtetem Graph von Startknoten s
- $O(n + k \cdot m)$ Knoten, $O(k \cdot m)$ Kanten

$\Rightarrow O(n + k \cdot m)$ Laufzeit



Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$ [ÜB 6]
Ges.:: Distanzen von Startknoten s

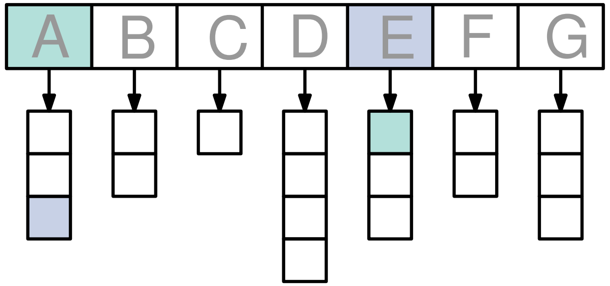


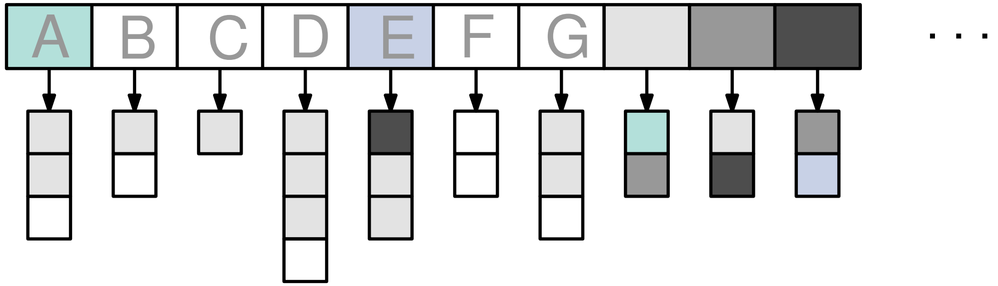
Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1
- BFS auf ungewichtetem Graph von Startknoten s
- $O(n + k \cdot m)$ Knoten, $O(k \cdot m)$ Kanten

$\Rightarrow O(n + k \cdot m)$ Laufzeit

Warum nicht einfach Dijkstra?



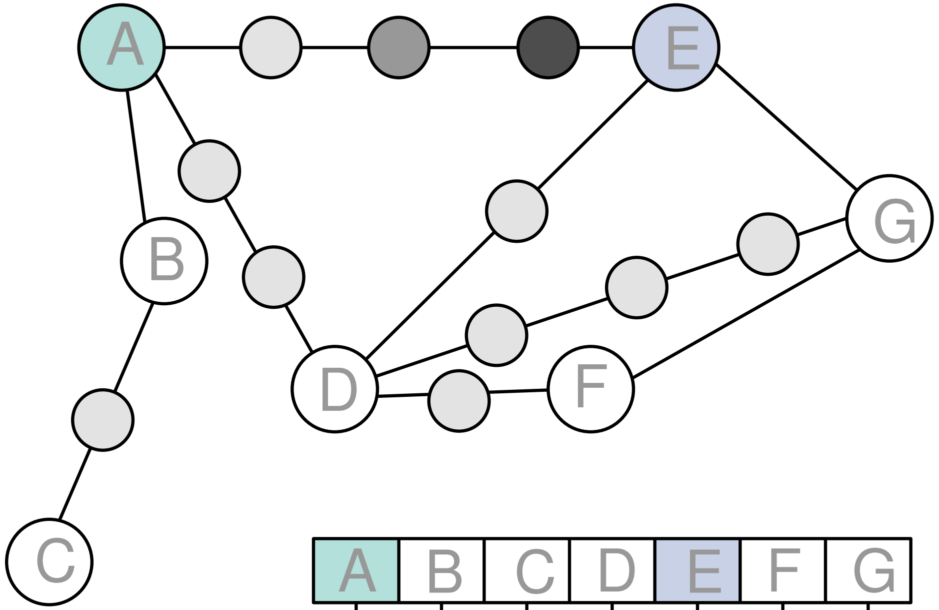


Beschränkte Kantengewichte

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s

Warum ohne 0?

[ÜB 6]

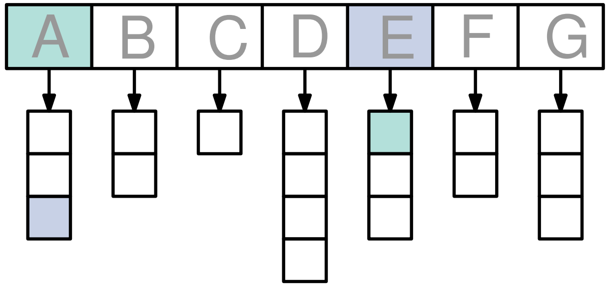


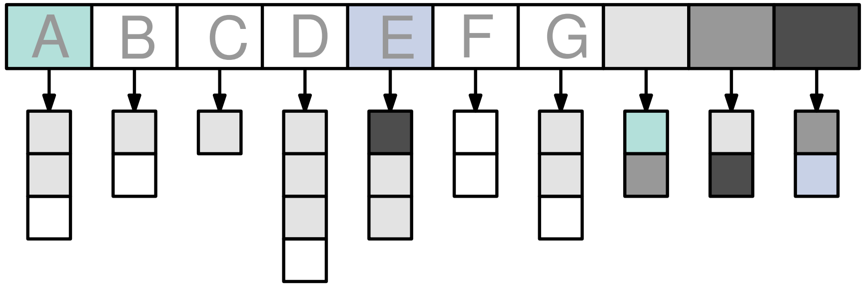
Lösung

- Kante mit Gewicht $w \Rightarrow w$ Kanten mit Gewicht 1
- BFS auf ungewichtetem Graph von Startknoten s
- $O(n + k \cdot m)$ Knoten, $O(k \cdot m)$ Kanten

$\Rightarrow O(n + k \cdot m)$ Laufzeit

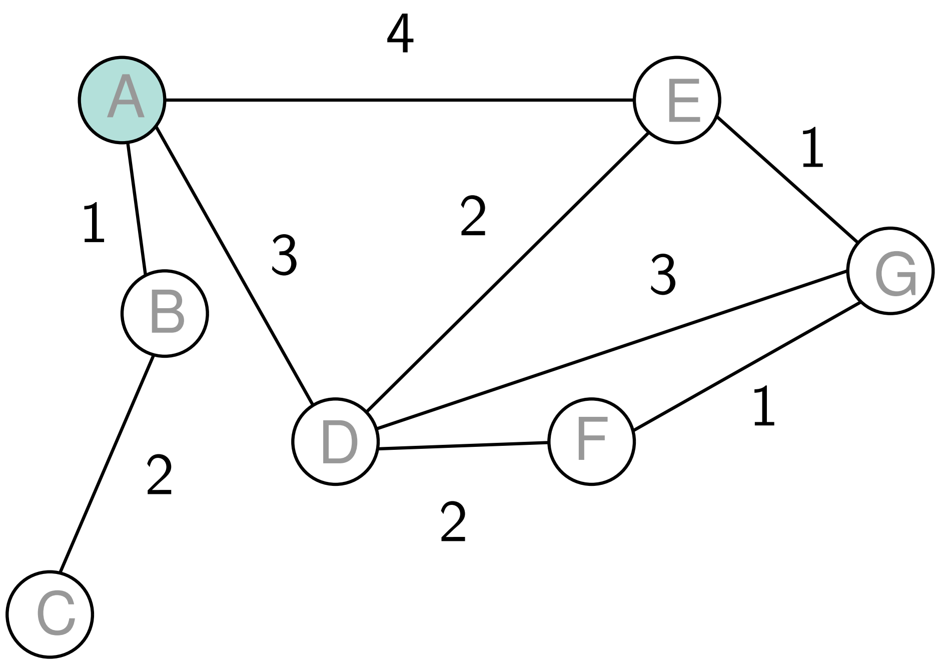
Warum nicht einfach Dijkstra?





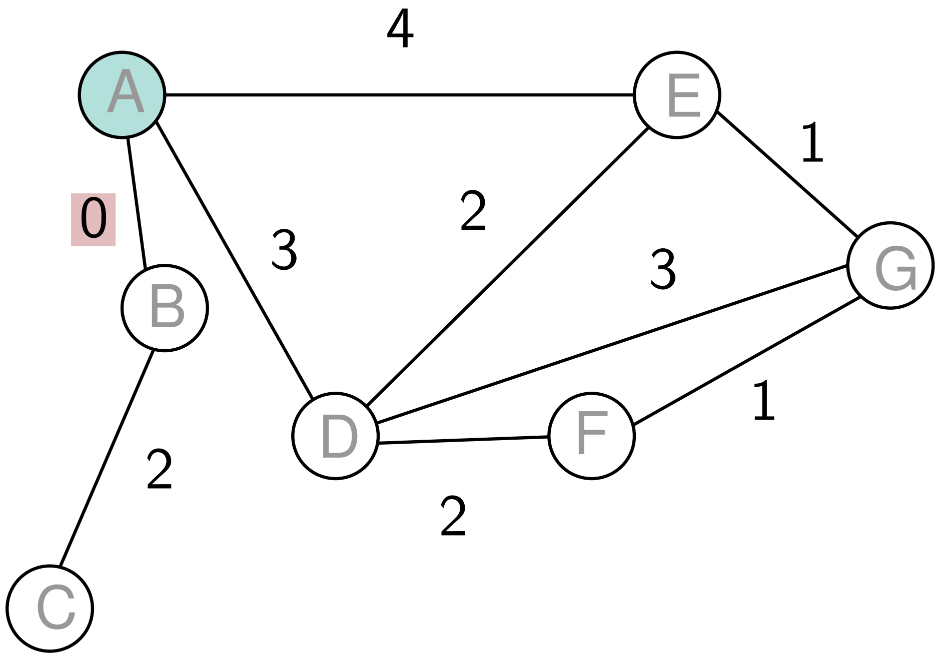
Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



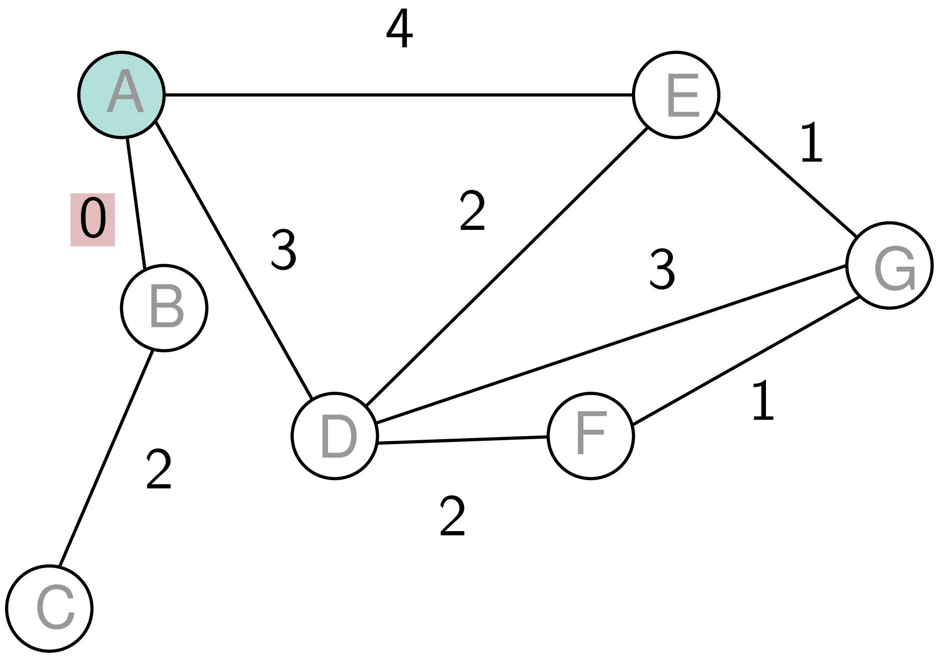
Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



Beschränkte Kantengewichte – mit 0

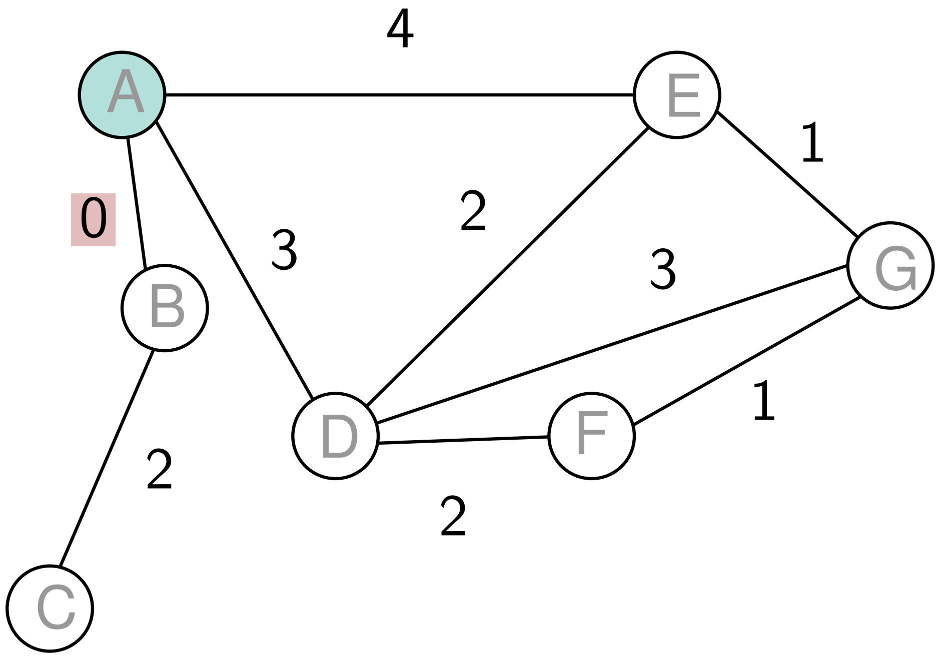
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt

Beschränkte Kantengewichte – mit 0

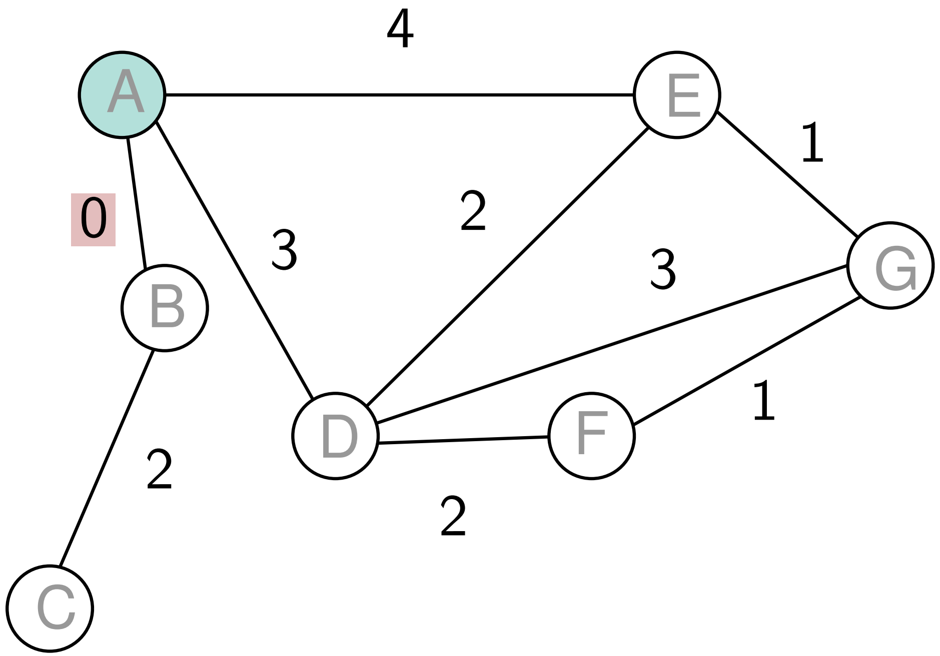
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s

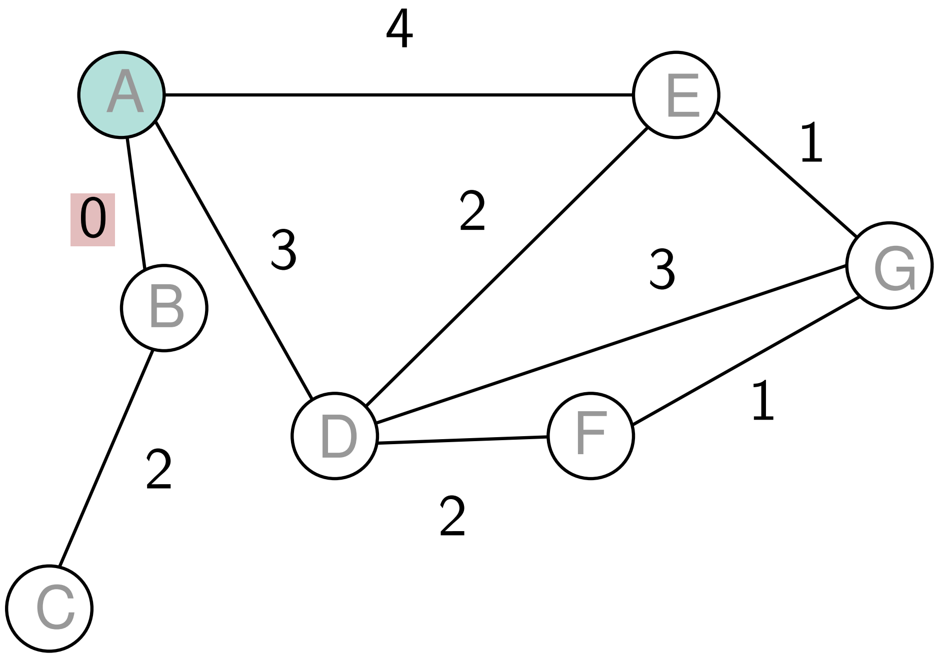


- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Wie?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



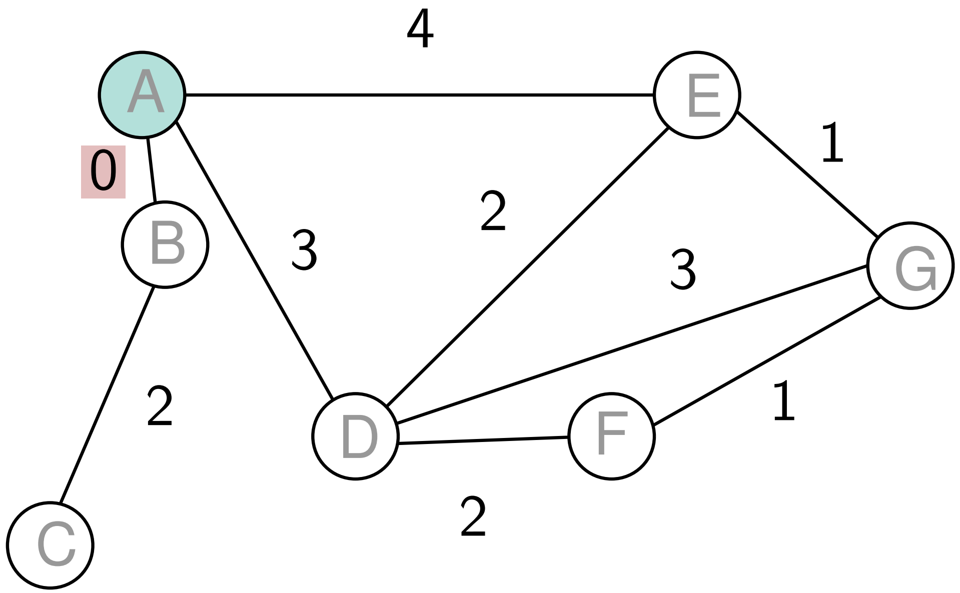
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Wie?

- kontrahiere nacheinander Kanten mit Gewicht 0

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



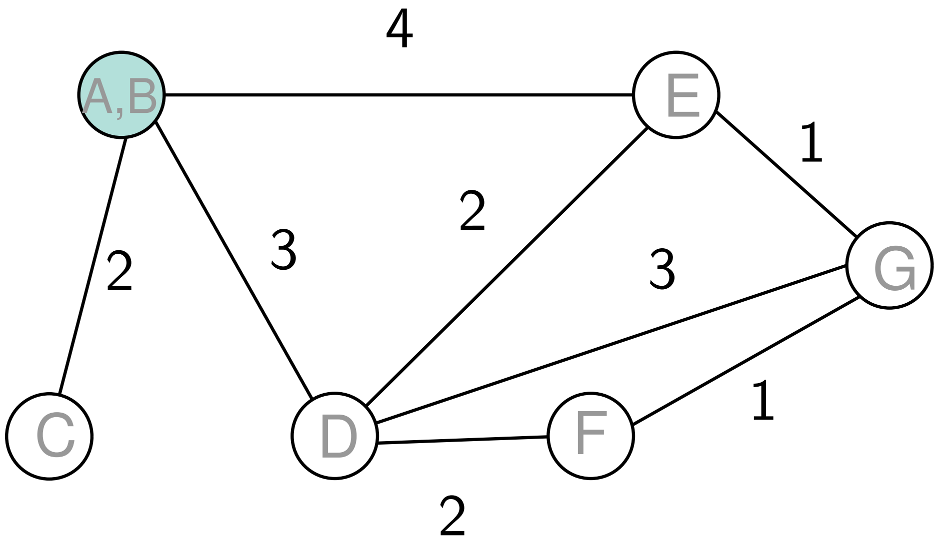
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Wie?

- kontrahiere nacheinander Kanten mit Gewicht 0

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



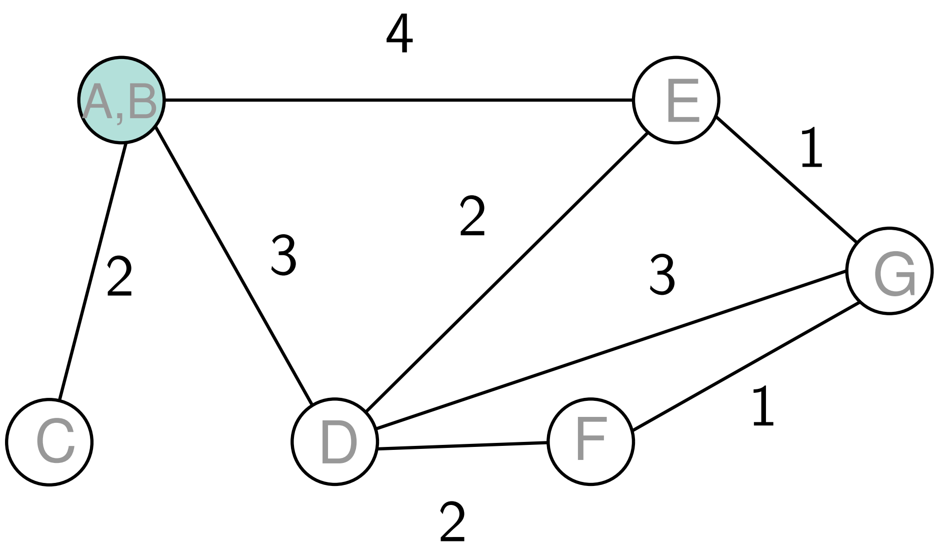
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Wie?

- kontrahiere nacheinander Kanten mit Gewicht 0

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s

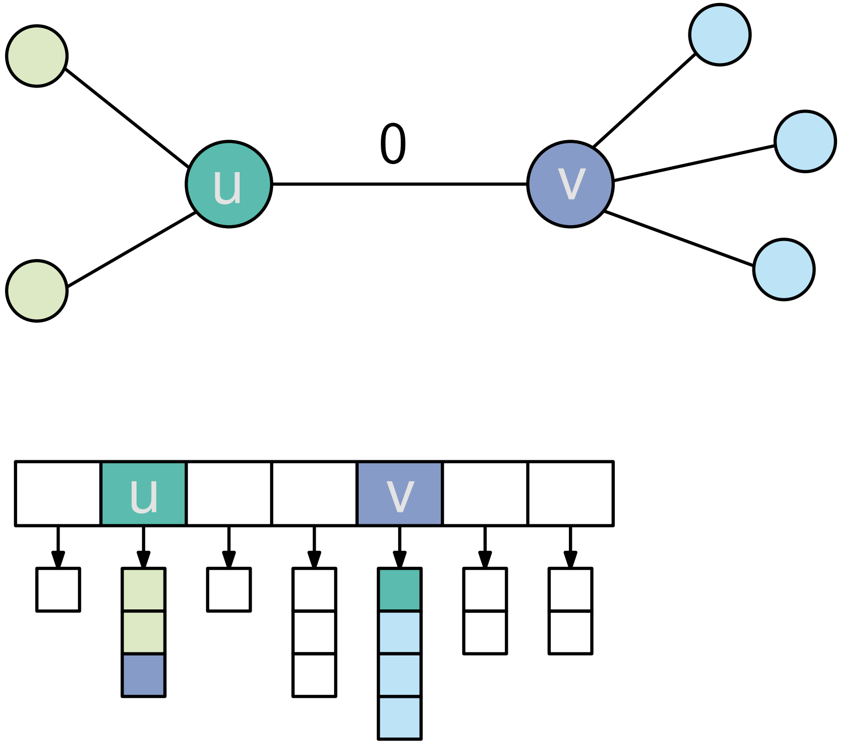


- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Wie?

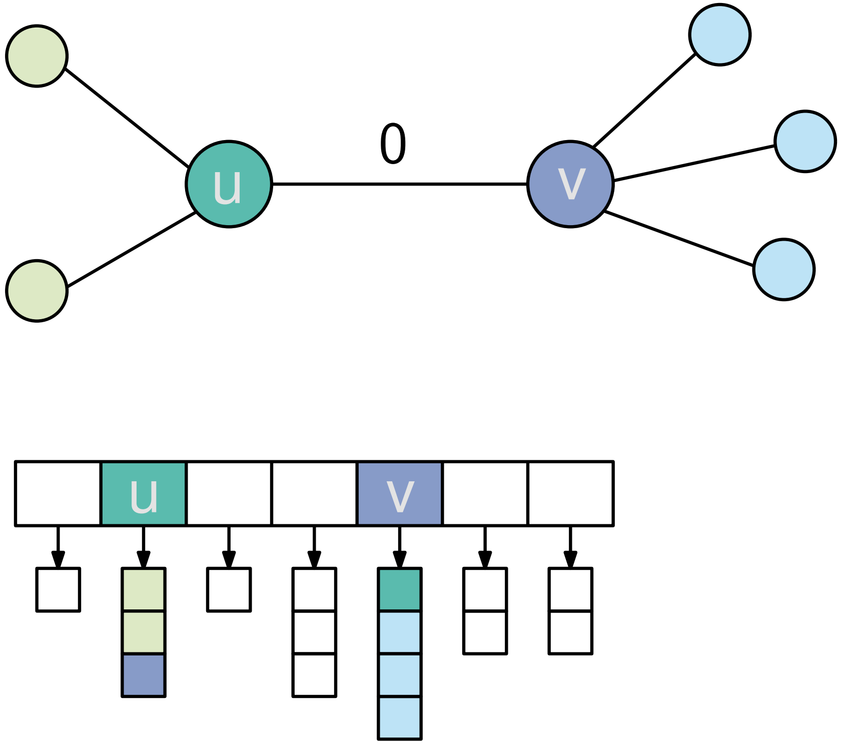
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

Einschub: Kantenkontraktion

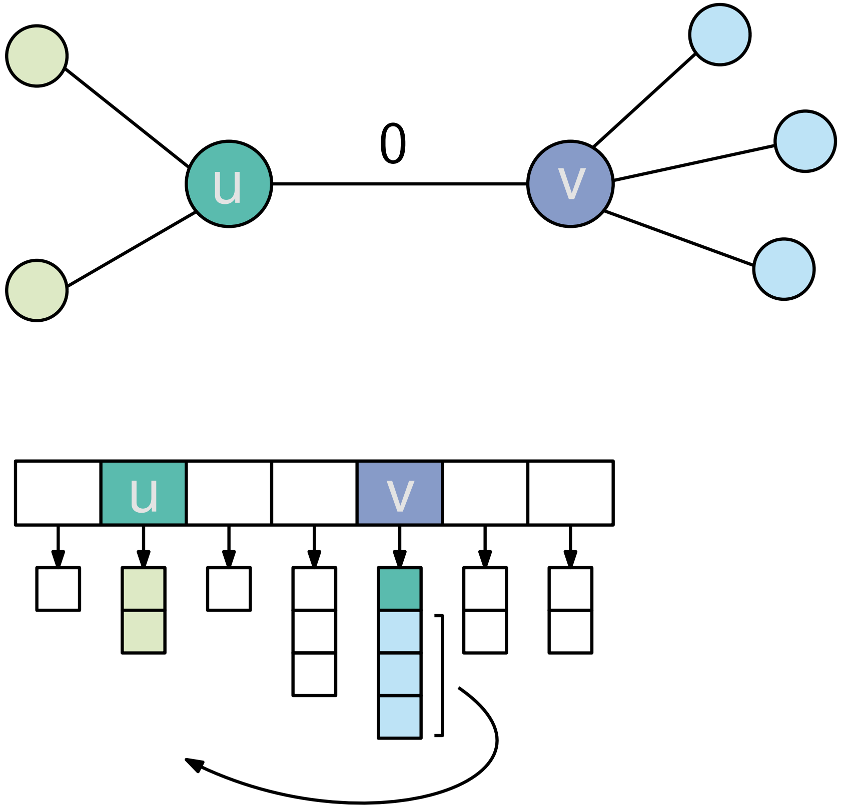


Einschub: Kantenkontraktion

- nicht für jede Kontraktion eine neue Adjazenzliste

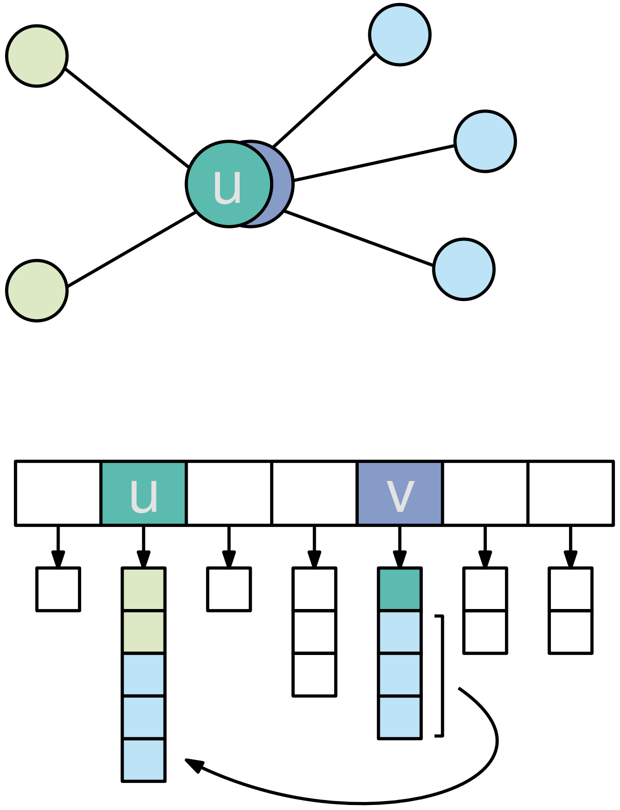


Einschub: Kantenkontraktion



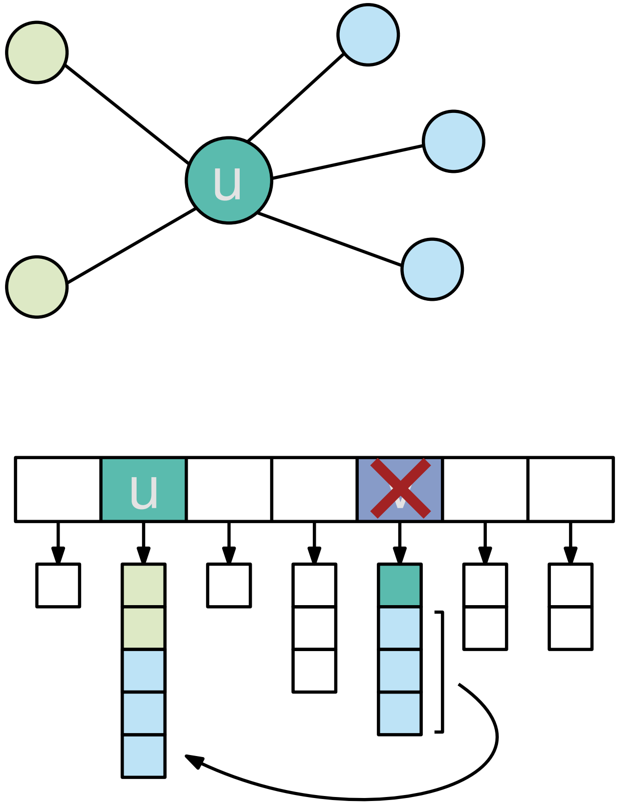
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu

Einschub: Kantenkontraktion



- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu

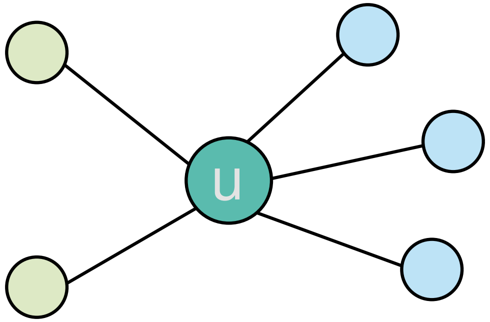
Einschub: Kantenkontraktion

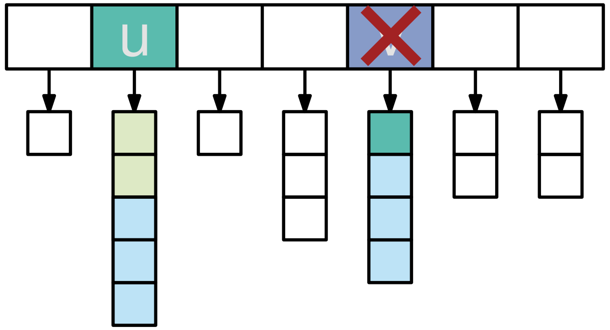


- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v

Einschub: Kantenkontraktion

Karlsruher Institut für Technologie

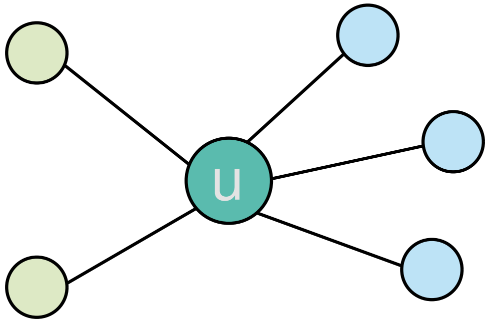


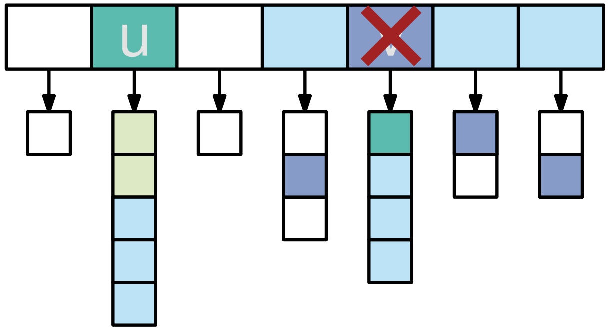


- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v

Sind wir fertig?

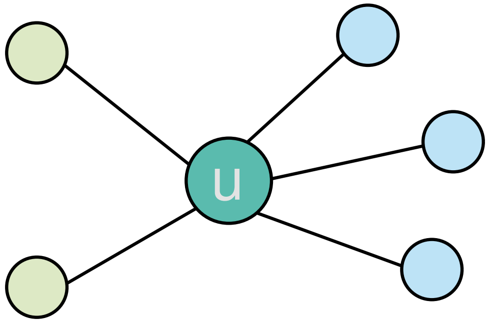
Einschub: Kantenkontraktion

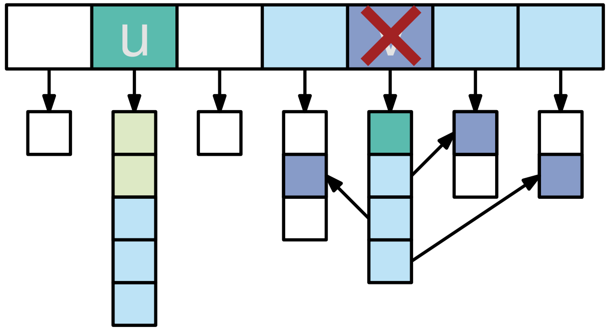




- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten

Einschub: Kantenkontraktion

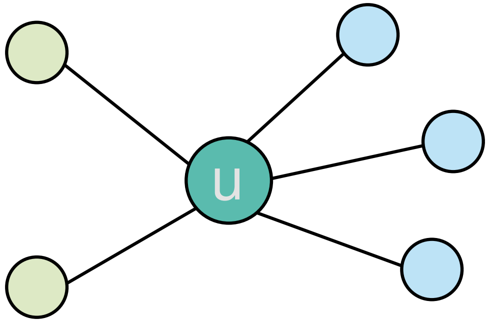


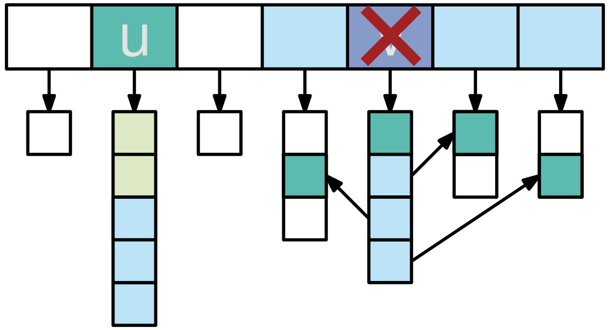


- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten

siehe Übung 3

Einschub: Kantenkontraktion

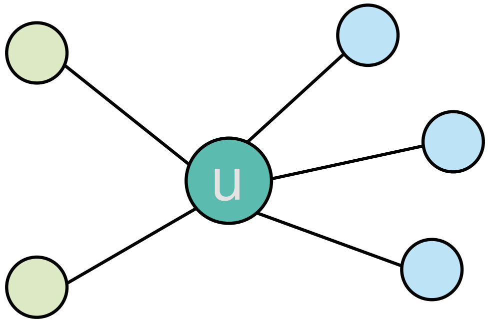


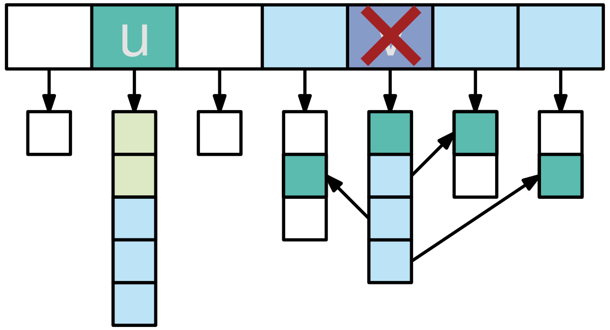


- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten

siehe Übung 3

Einschub: Kantenkontraktion



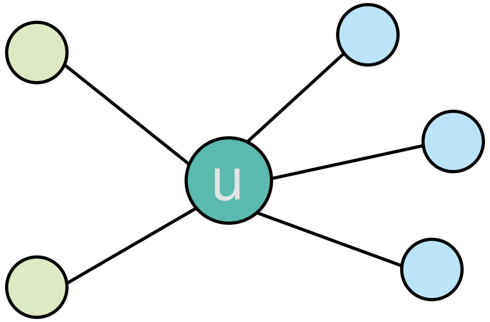


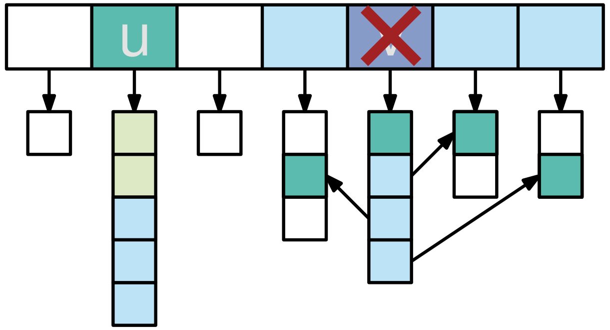
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten

siehe Übung 3

Sind wir fertig?

Einschub: Kantenkontraktion





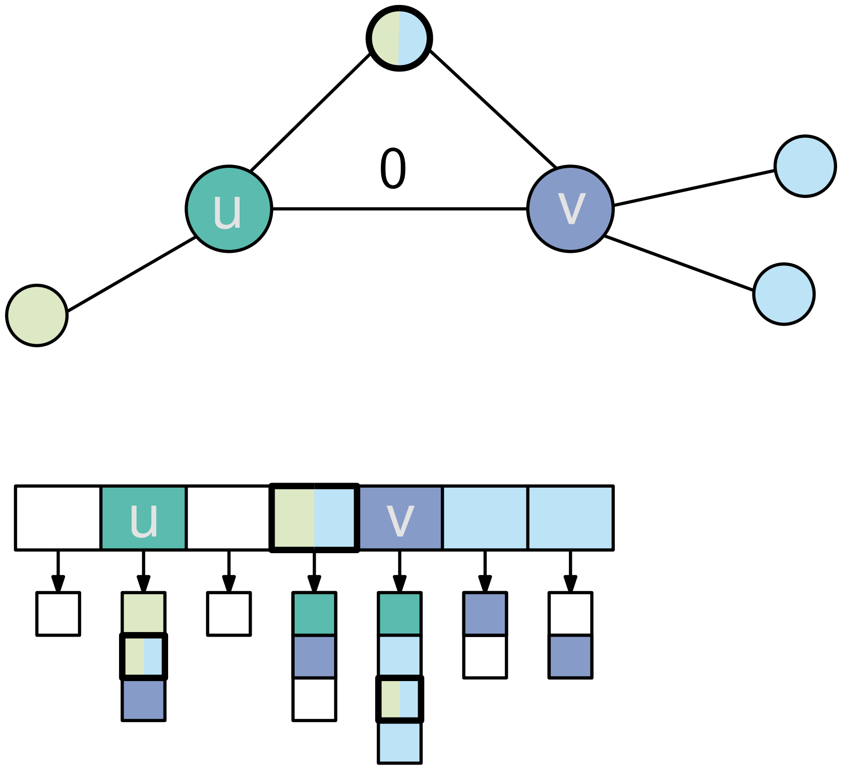
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten

siehe Übung 3

u und v können sich Nachbarn teilen!

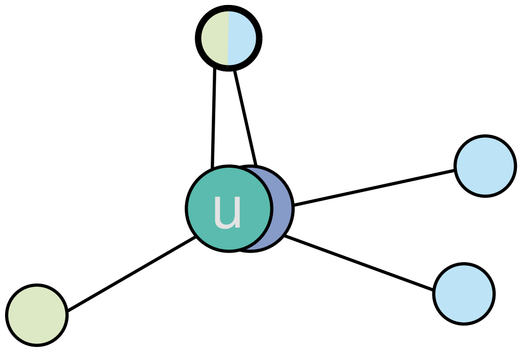
Einschub: Kantenkontraktion

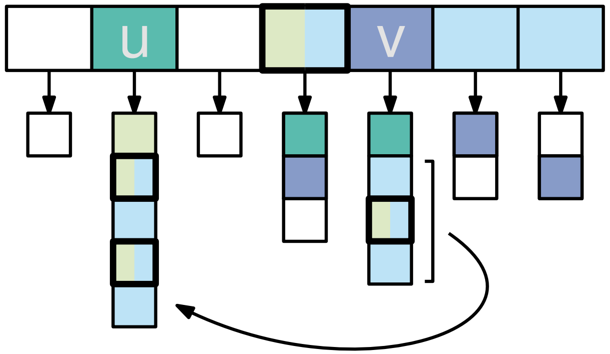
- nicht für jede Kontraktion eine neue Adjazenzliste



u und v können sich Nachbarn teilen!

Einschub: Kantenkontraktion

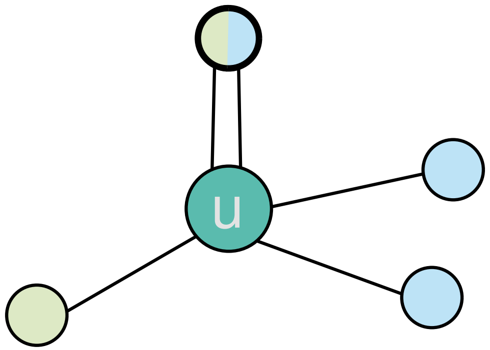




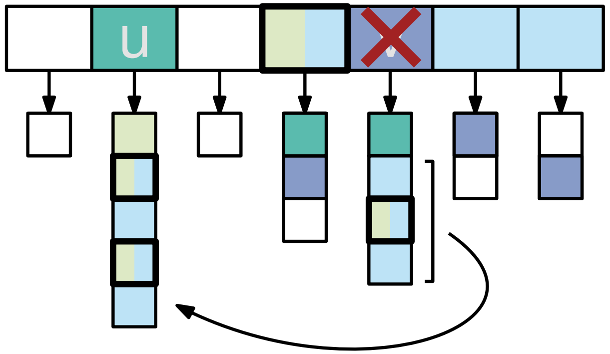
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu

u und v können sich Nachbarn teilen!

Einschub: Kantenkontraktion

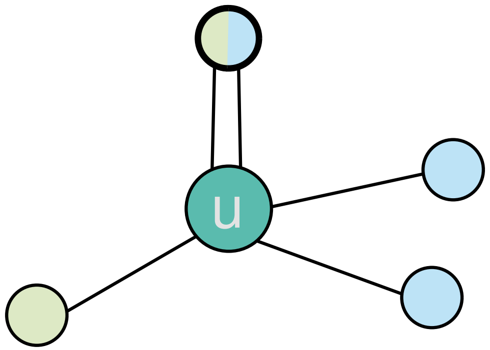


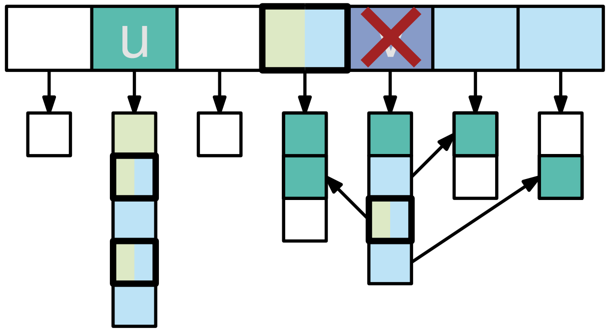
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v



u und v können sich Nachbarn teilen!

Einschub: Kantenkontraktion



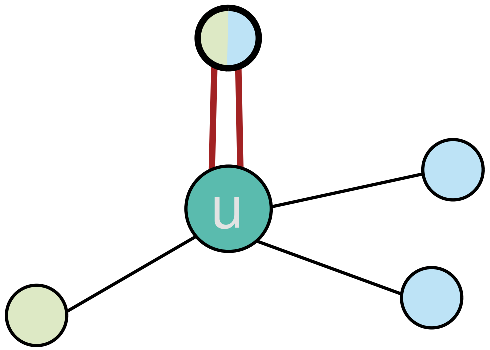


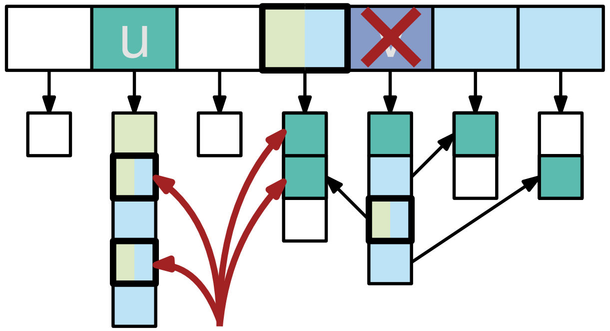
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten

siehe Übung 3

u und v können sich Nachbarn teilen!

Einschub: Kantenkontraktion



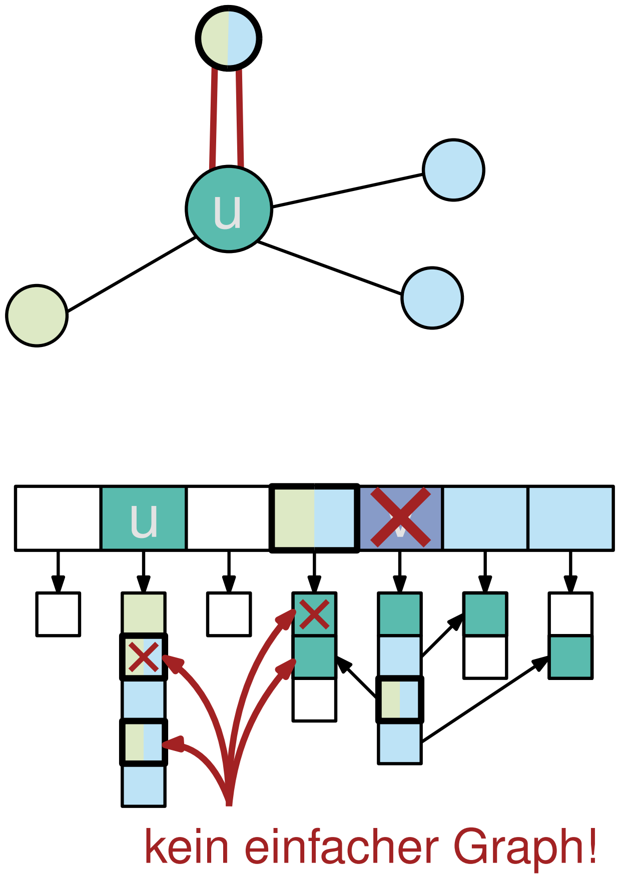


- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten

siehe Übung 3

u und v können sich Nachbarn teilen!

Einschub: Kantenkontraktion

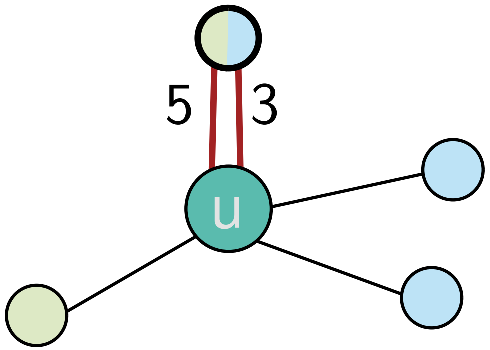


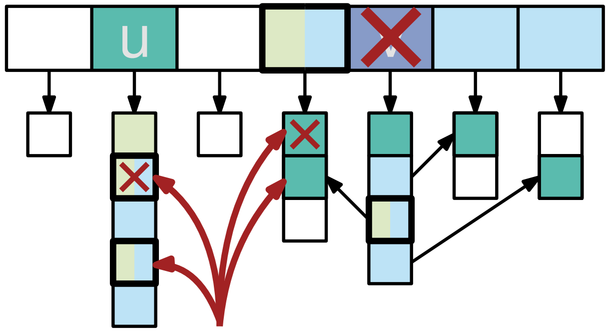
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten
- lösche Duplikate

siehe Übung 3

u und v können sich Nachbarn teilen!

Einschub: Kantenkontraktion





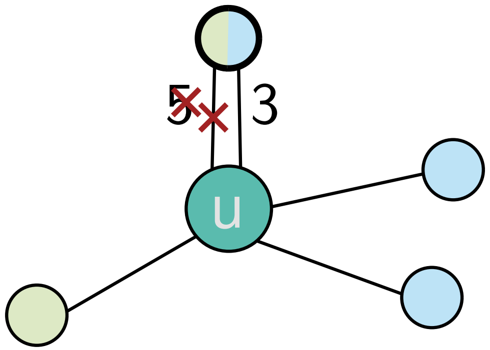
- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten
- lösche Duplikate

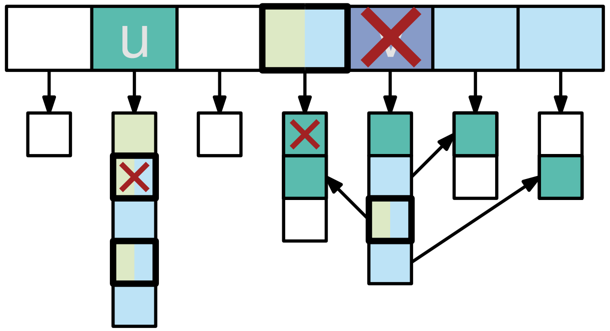
siehe Übung 3

u und v können sich Nachbarn teilen!

Kantengewicht der Kante, wenn Distanz gleich bleiben soll?
- $\min(c(e_1), c(e_2))$
- $\max(c(e_1), c(e_2))$
- $c(e_1) + c(e_2)$

Einschub: Kantenkontraktion





- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten
- lösche Duplikate

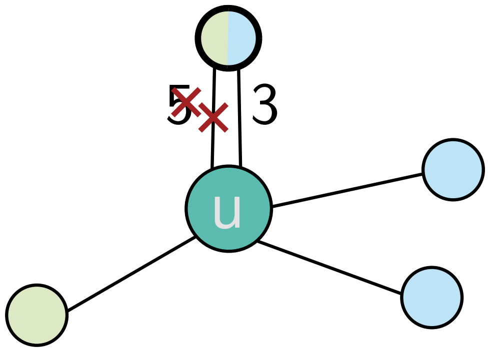
siehe Übung 3

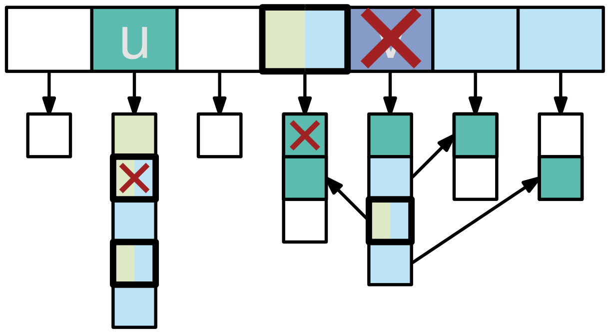
u und v können sich Nachbarn teilen!

Kantengewicht der Kante, wenn Distanz gleich bleiben soll?

- $\min(c(e_1), c(e_2))$
- $\max(c(e_1), c(e_2))$
- $c(e_1) + c(e_2)$

Einschub: Kantenkontraktion





- nicht für jede Kontraktion eine neue Adjazenzliste
- lösche v aus Liste von u
- füge alle Nachbarn von v zu u hinzu
- lösche oder markiere v
- Kanten von $N(v)$ zu v updaten
- lösche Duplikate

siehe Übung 3

Laufzeit: $|N(u)|\log(|N(u)|) + |N(v)|\log(|N(v)|) + \sum_{w \in N(v)} |N(w)|$

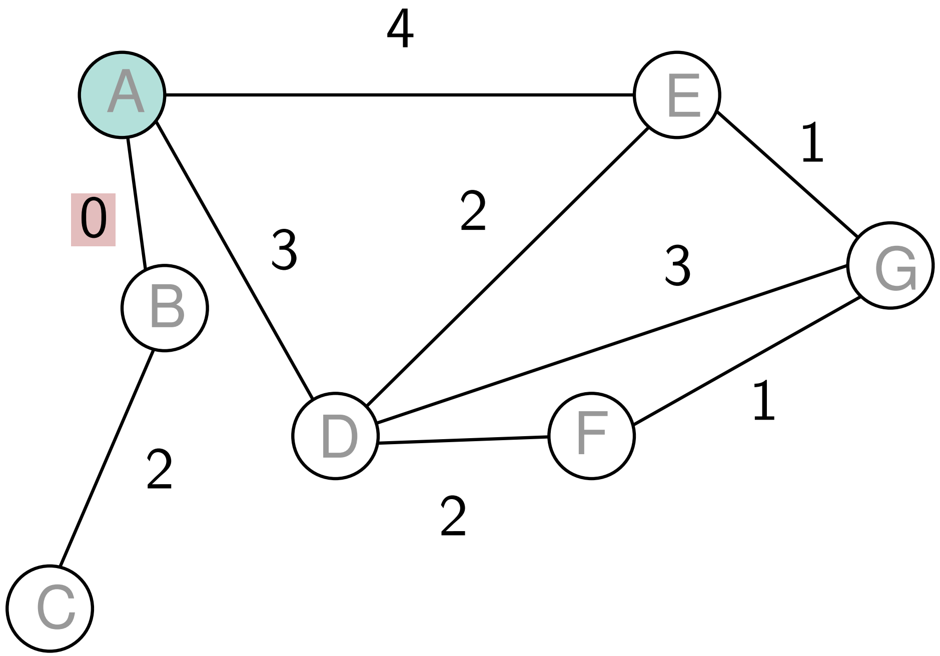
u und v können sich Nachbarn teilen!

Kantengewicht der Kante, wenn Distanz gleich bleiben soll?

- $\min(c(e_1), c(e_2))$
- $\max(c(e_1), c(e_2))$
- $c(e_1) + c(e_2)$

Beschränkte Kantengewichte – mit 0

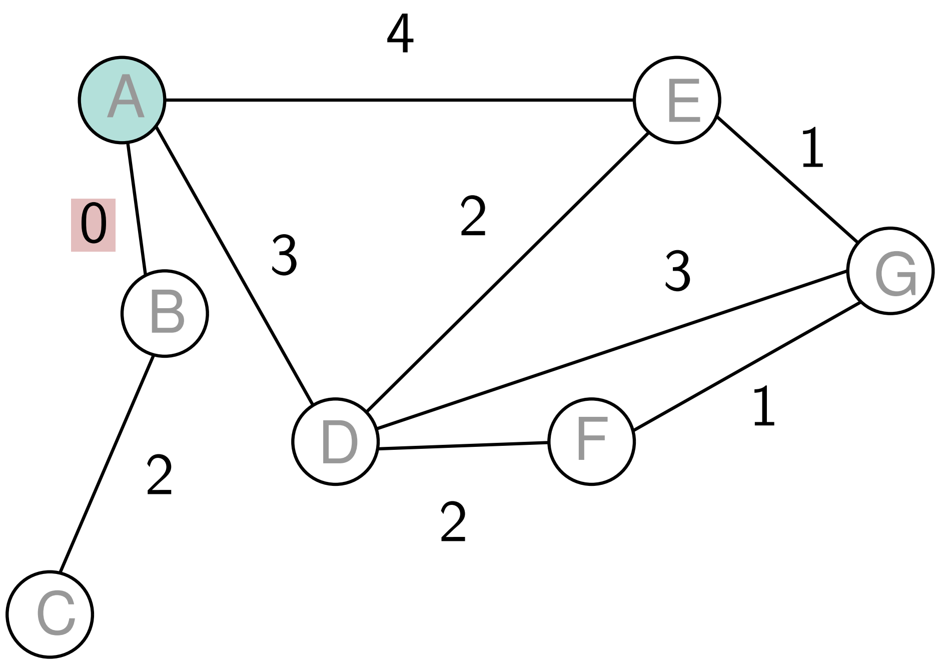
Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s

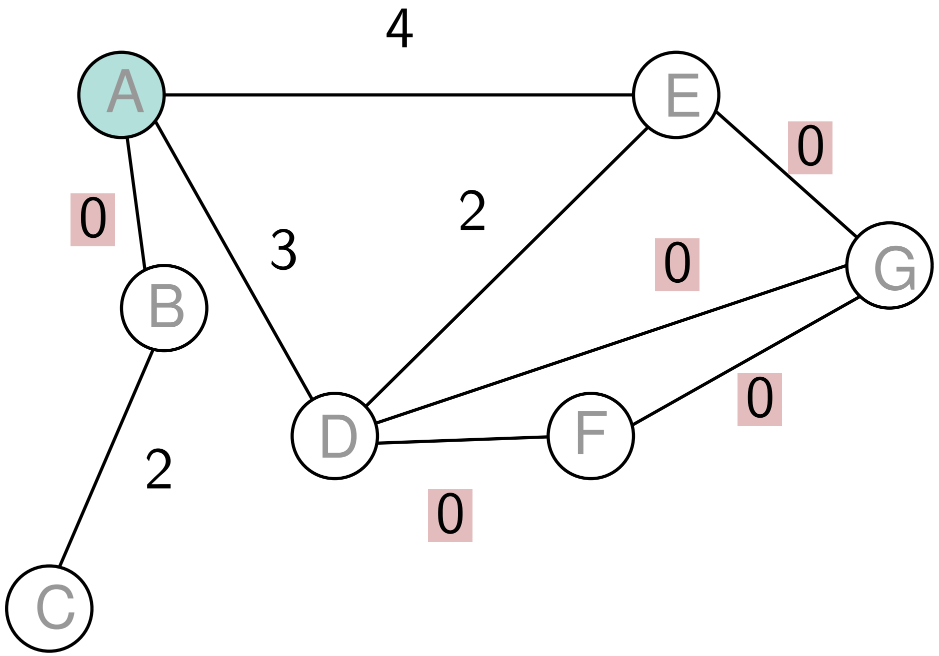


- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

↯ teuer und kompliziert

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s

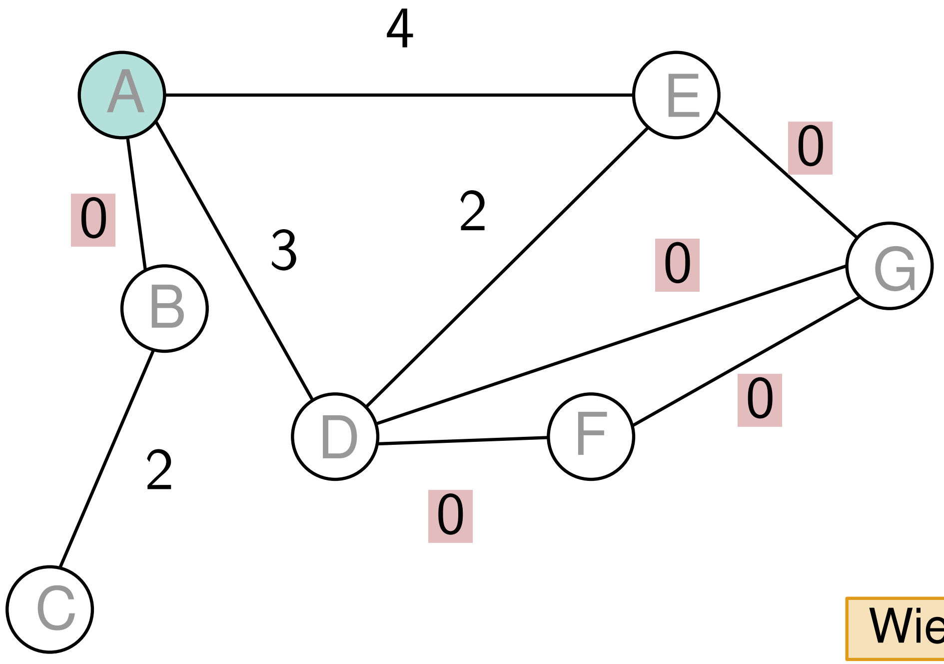


- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

$\lightning$ teuer und kompliziert

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



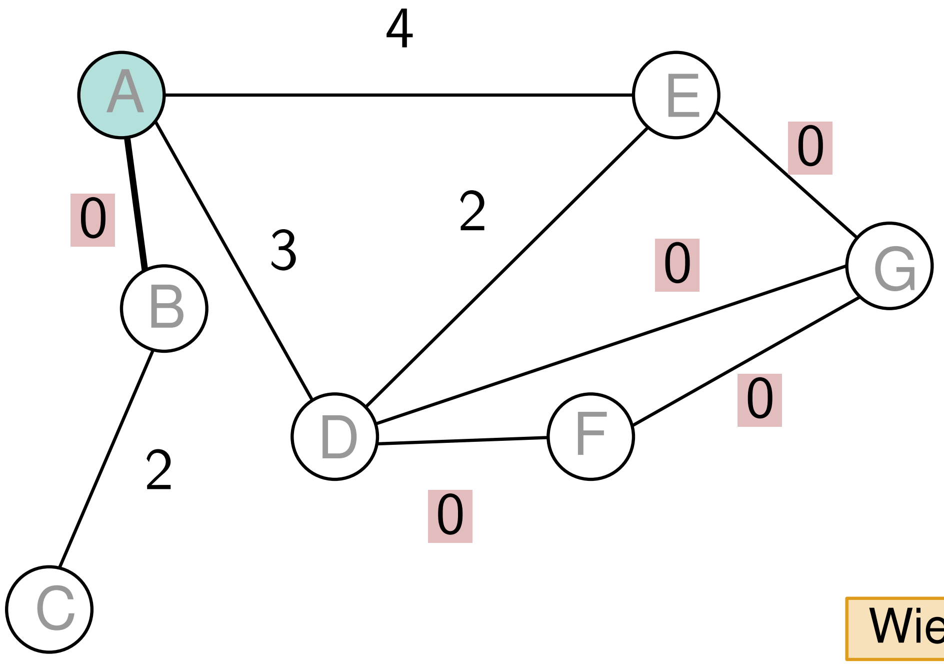
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

↯ teuer und kompliziert

Wie viele Knoten und Kanten sind übrig nach den Kontraktionen?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



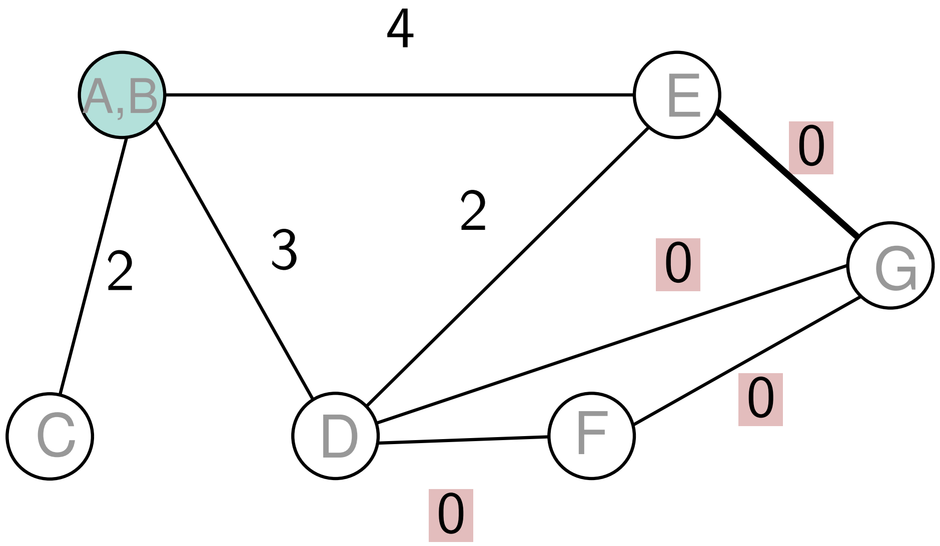
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

↯ teuer und kompliziert

Wie viele Knoten und Kanten sind übrig nach den Kontraktionen?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



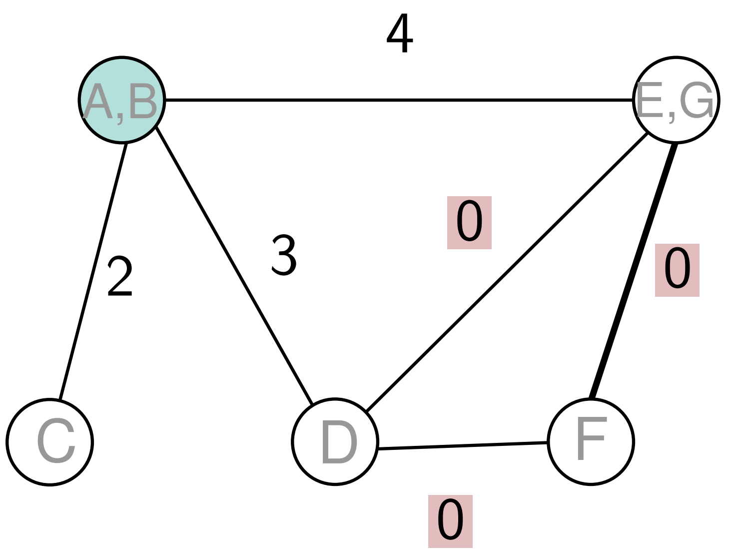
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

$\lightning$ teuer und kompliziert

Wie viele Knoten und Kanten sind übrig nach den Kontraktionen?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



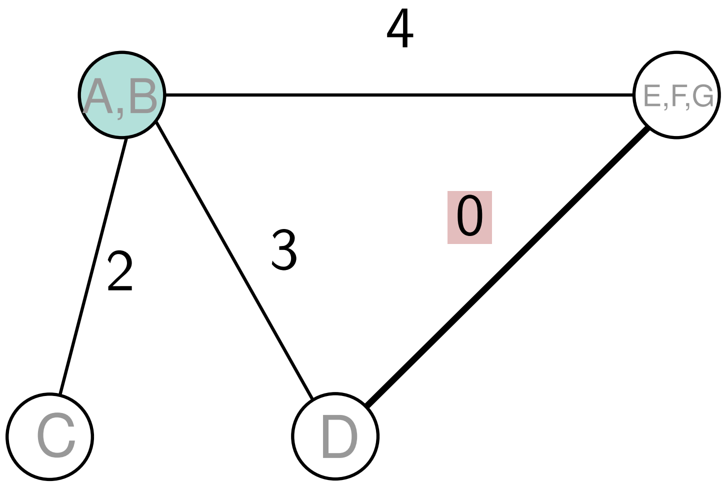
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

$\lightning$ teuer und kompliziert

Wie viele Knoten und Kanten sind übrig nach den Kontraktionen?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



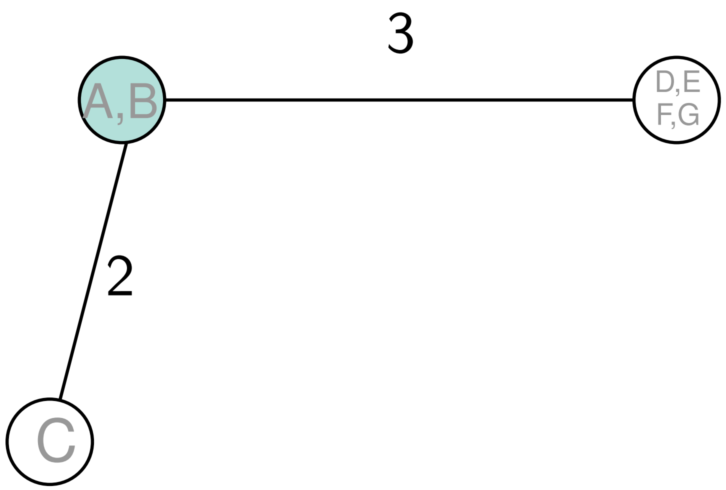
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

↯ teuer und kompliziert

Wie viele Knoten und Kanten sind übrig nach den Kontraktionen?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



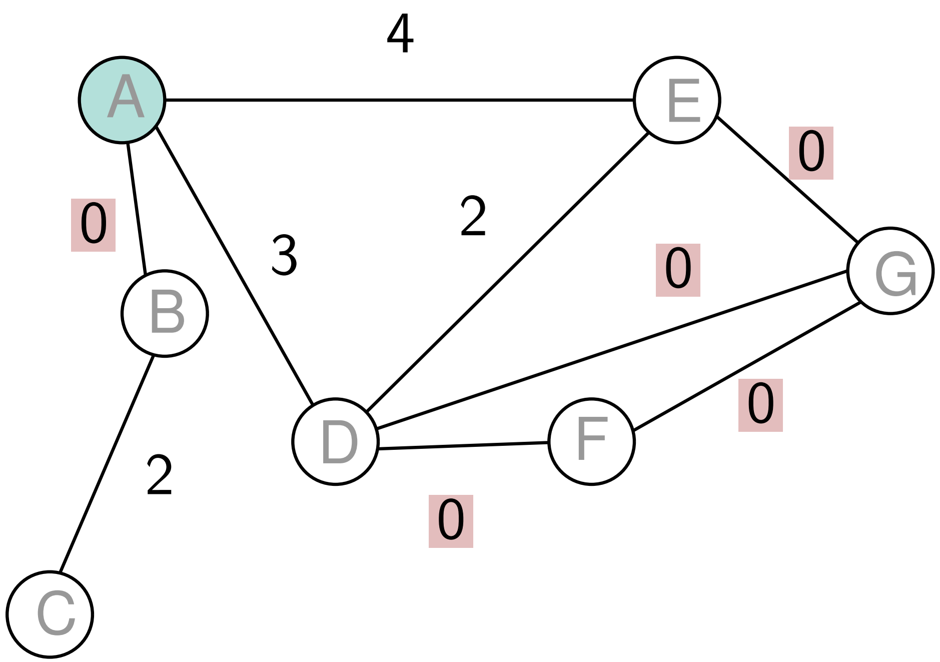
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten
- kontrahiere nacheinander Kanten mit Gewicht 0
- verwende den Algorithmus von vorhin

↯ teuer und kompliziert

Wie viele Knoten und Kanten sind übrig nach den Kontraktionen?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



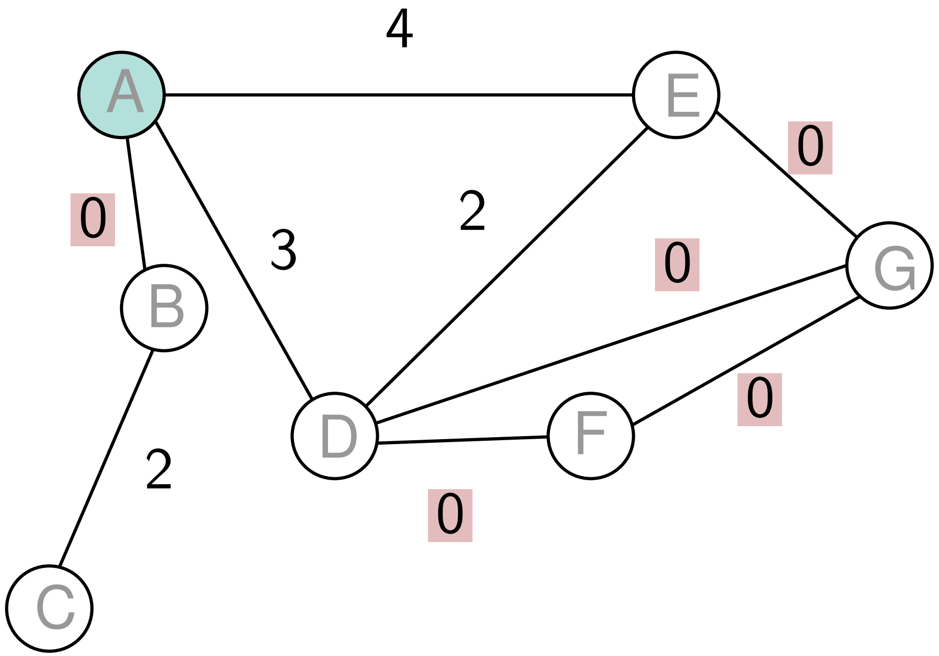
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

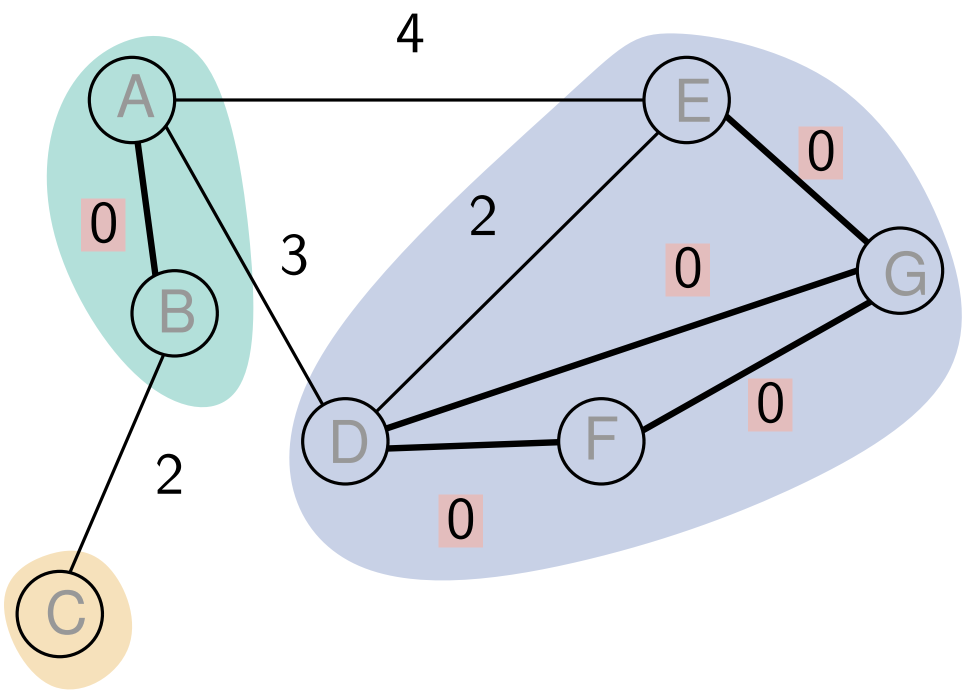
Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal

Welche Knoten fallen zusammen? Wie finden wir die?

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



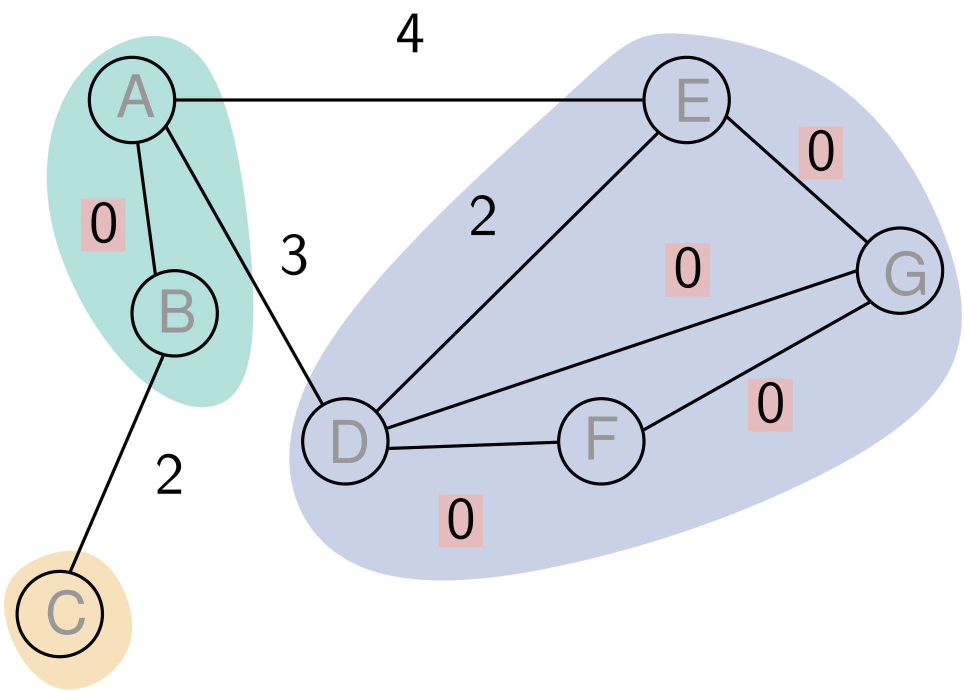
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



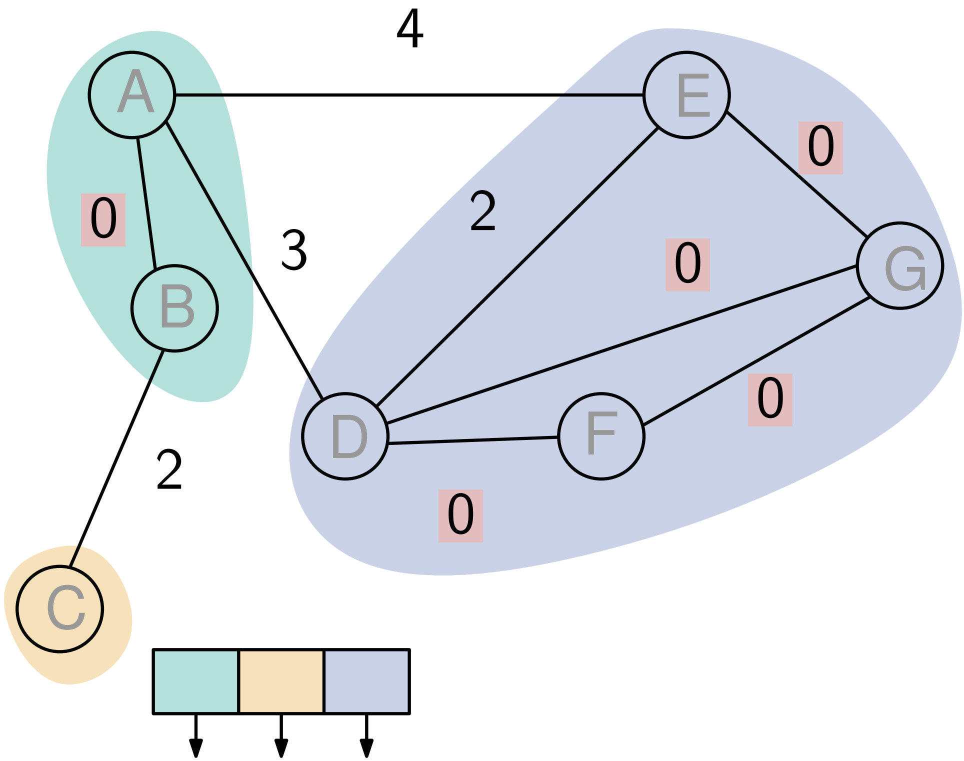
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



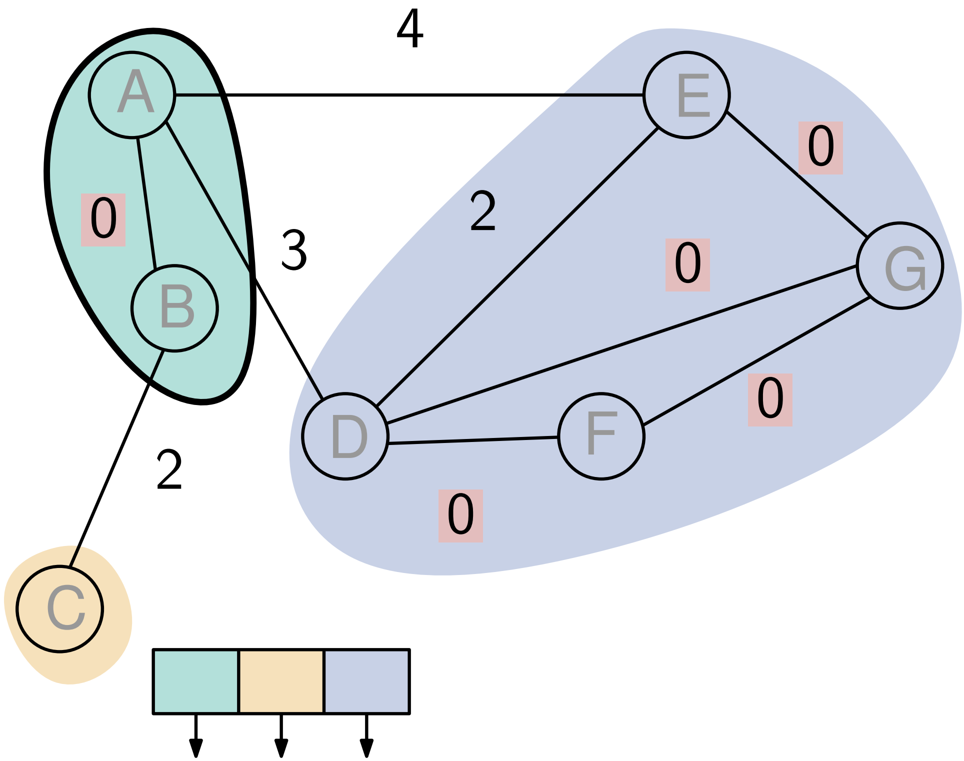
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



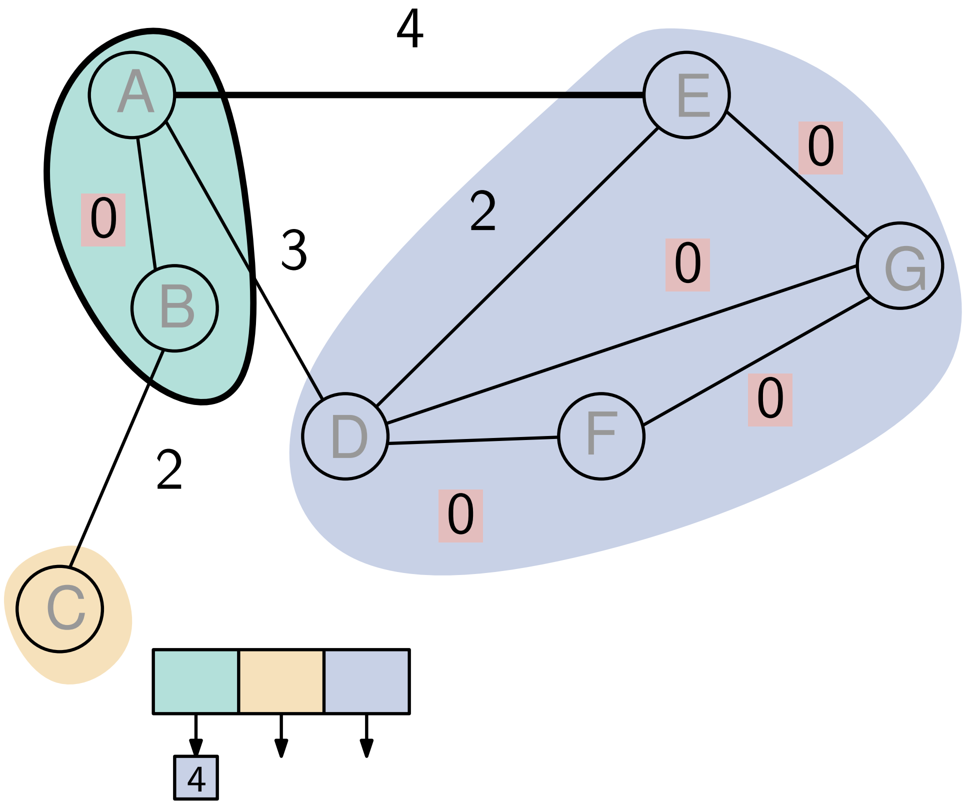
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



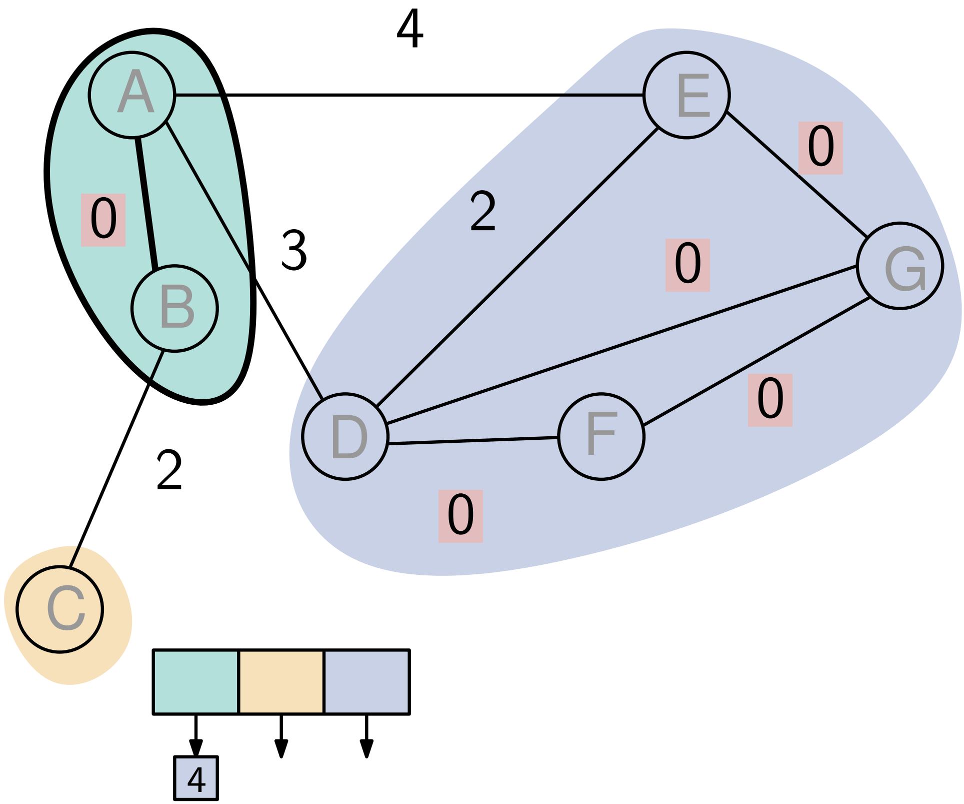
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



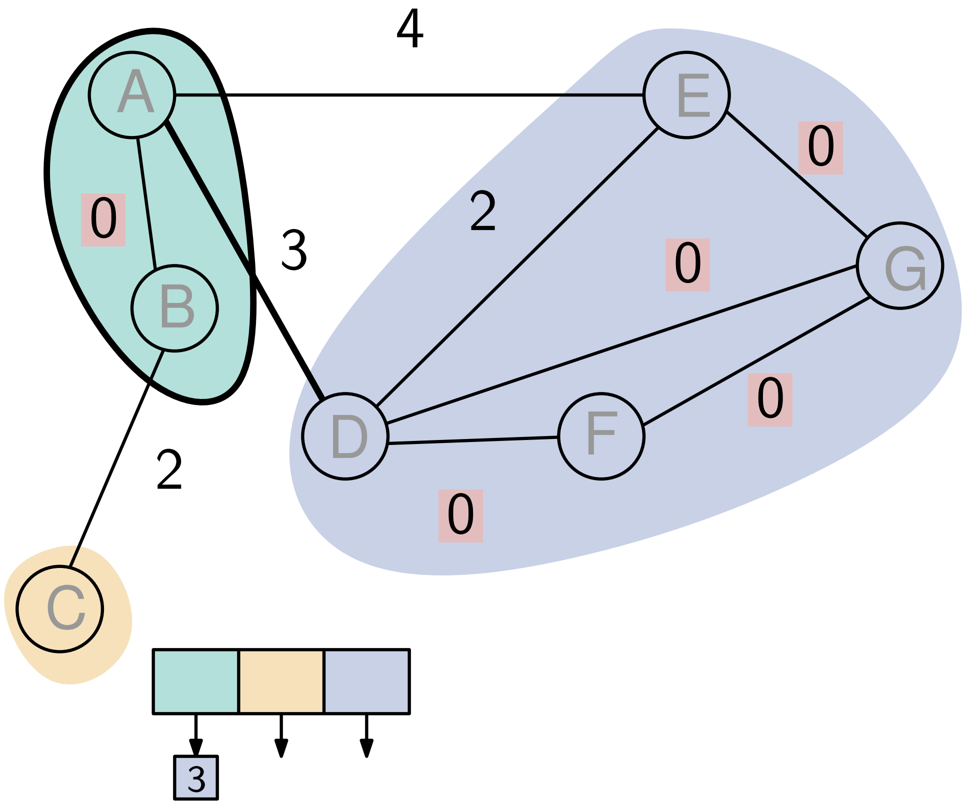
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



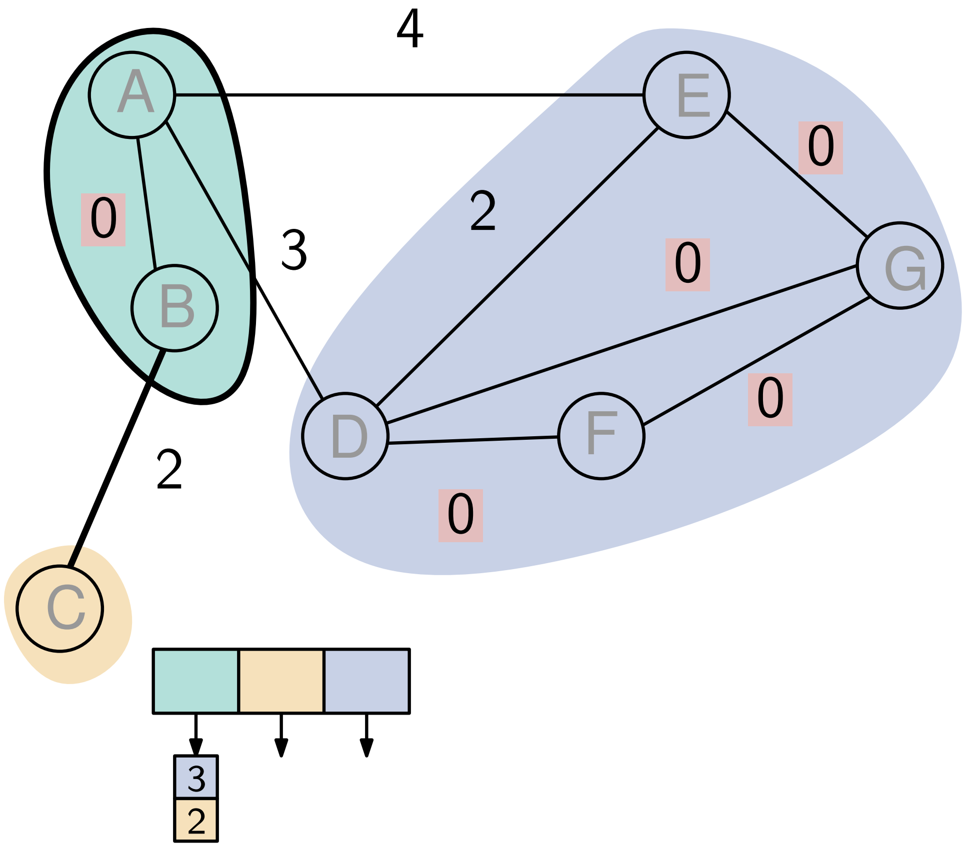
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



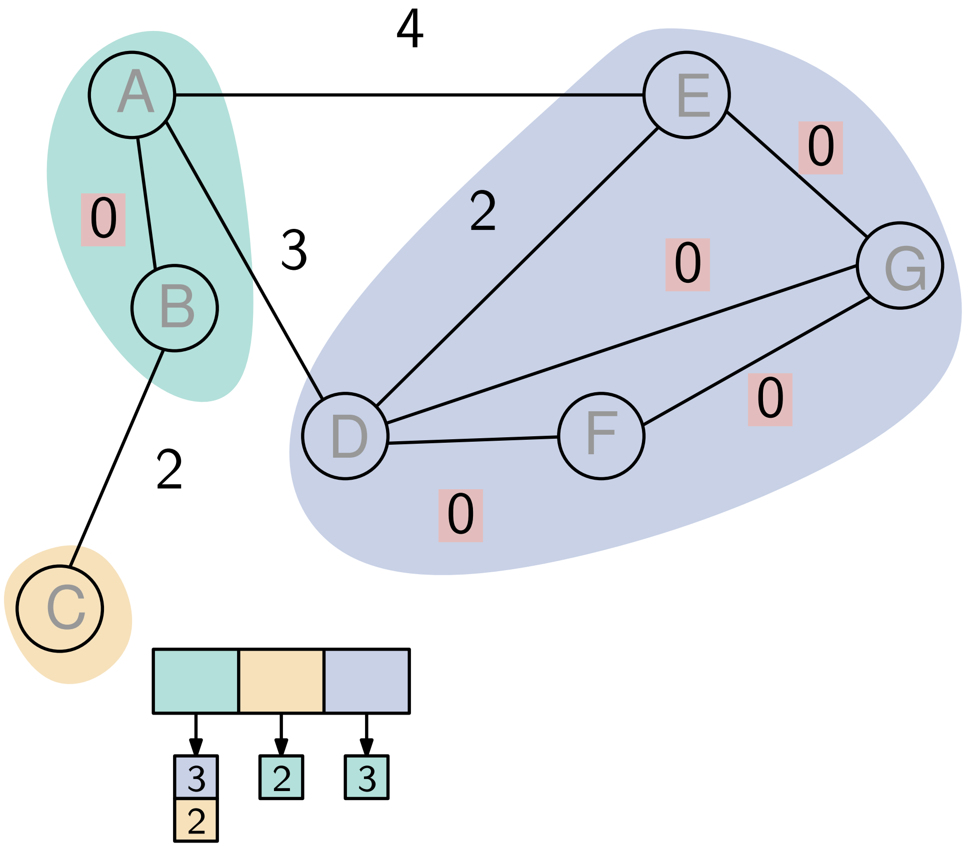
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



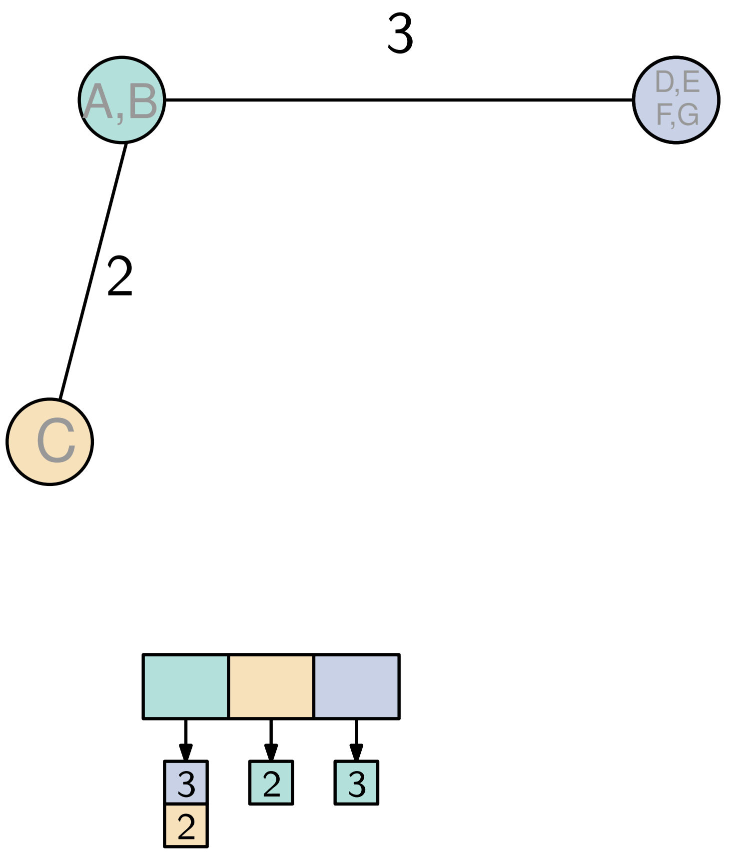
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



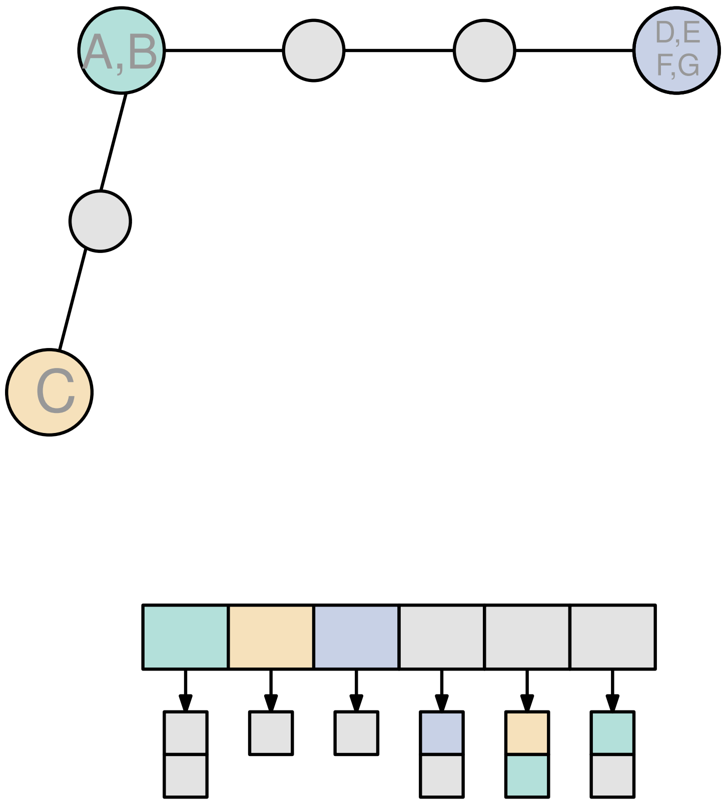
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf
- verwende den Algorithmus von vorhin

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



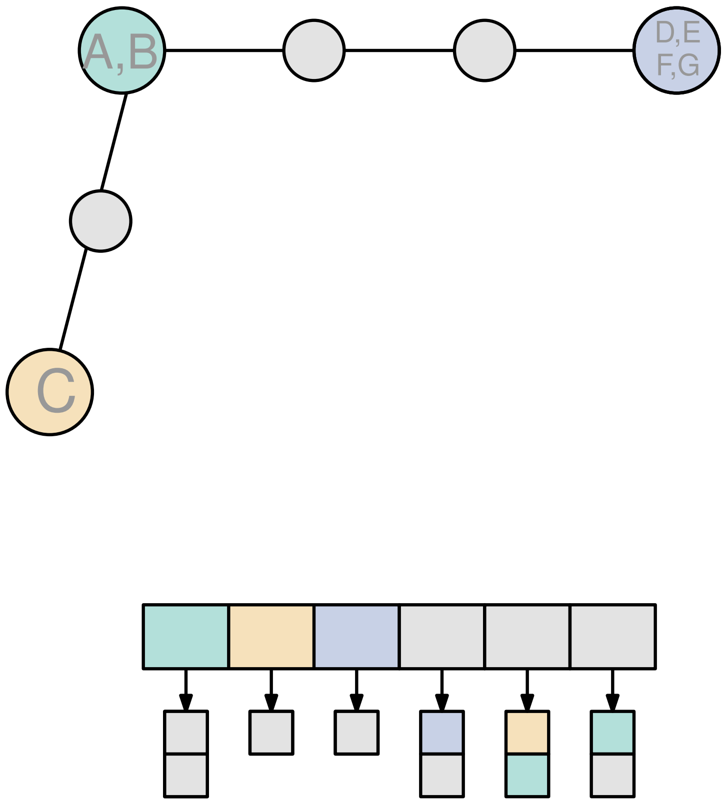
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- finde mit BFS 0-Zusammenhangskomponenten
- baue den kontrahierten Graphen neu auf
- verwende den Algorithmus von vorhin

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s



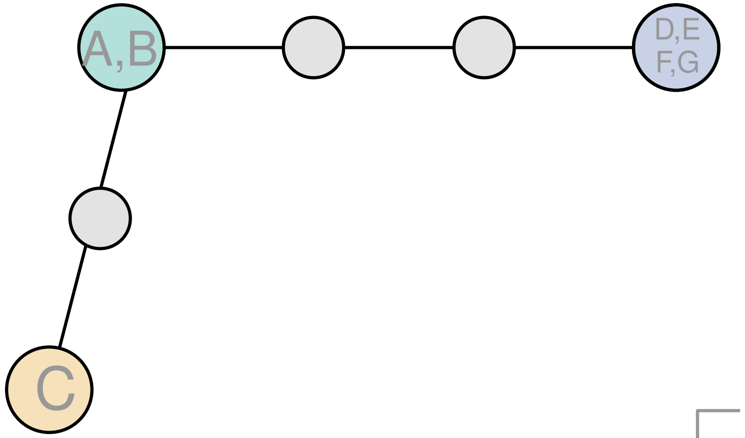
- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

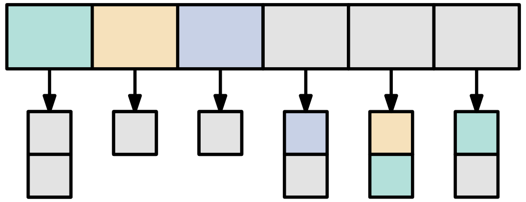
Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- $O(n+m)$ finde mit BFS 0-Zusammenhangskomponenten
- $O(n+m)$ baue den kontrahierten Graphen neu auf
- $O(n+km)$ verwende den Algorithmus von vorhin
- Laufzeit: $O(n + km)$

Beschränkte Kantengewichte – mit 0

Geg.: Graph mit natürlichen Kantengewichten in $(0, k]$
Ges.:: Distanzen von Startknoten s





- Kanten mit Gewicht 0 machen Algorithmus von vorher kaputt
- **Idee:** Kanten mit Gewicht 0 vorverarbeiten

Versuch 2

- kontrahiere alle Kanten mit Gewicht 0 auf einmal
- $O(n+m)$ finde mit BFS 0-Zusammenhangskomponenten
- $O(n+m)$ baue den kontrahierten Graphen neu auf
- $O(n+km)$ verwende den Algorithmus von vorhin
- Laufzeit: $O(n+km)$

Details: gute Übung (etwas tricky)