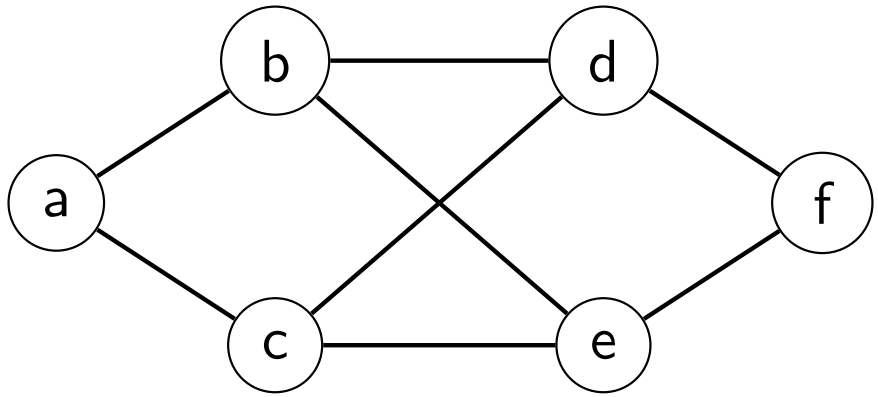
Folien aus der Vorlesung Optimierung I SS2013

Dr. Jens Maßberg
Institut für Optimierung und Operations Research,
Universität Ulm

July 10, 2013

- Datenstrukturen für Graphen und Digraphen
- Graph Scanning Algorithmus
- BFS-Bäume
- Minimum Spanning Trees
- Kruskal's Algorithm
- Prim's Algorithm
- Dijkstra's Algorithmus
- Moore-Bellman-Ford Algorithmus
- Floyd-Warshall Algorithmus

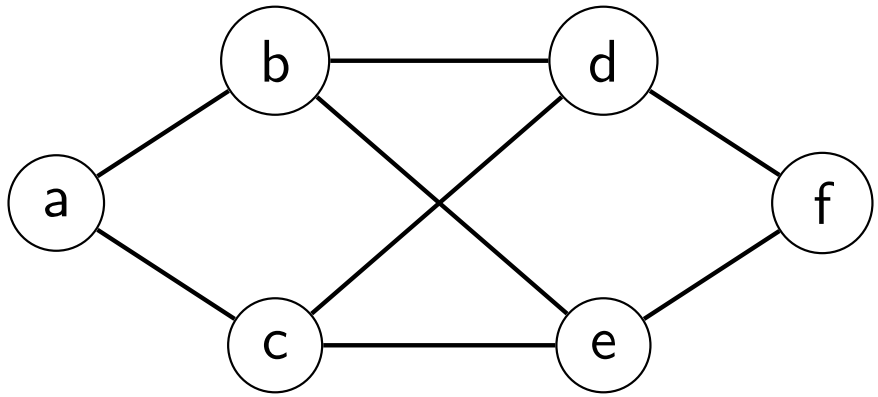
Inzidenzmatrix



	ab	ac	bd	be	cd	ce	df	ef
a	1	1						
b	1		1	1				
c		1			1	1		
d			1		1		1	
e				1		1		1
f							1	1

Speicheraufwand: $O(nm)$.

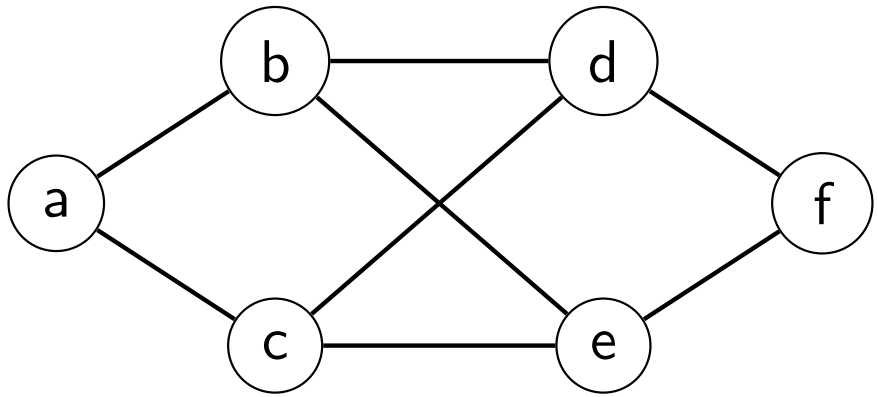
Adjazenzmatrix



	a	b	c	d	e	f
a		1	1			
b	1			1	1	
c	1		1	1		
d		1	1			1
e		1	1			1
f				1	1	

Speicheraufwand: $O(n^2)$.

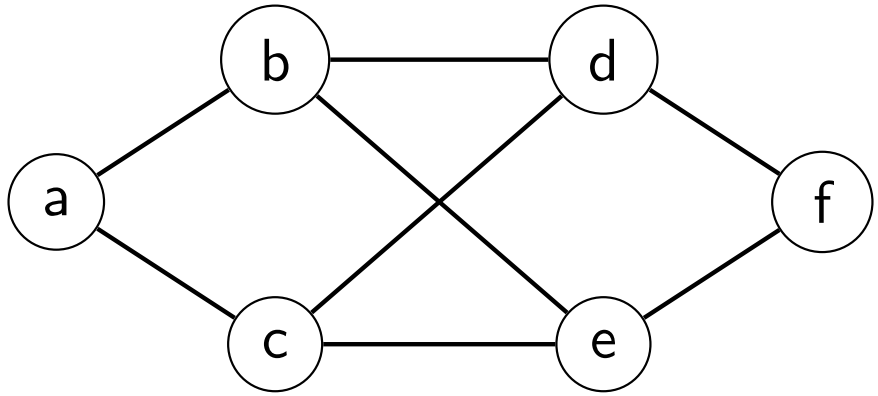
Kantenliste



$(a, b), (a, c), (b, d), (b, e)(c, d), (c, e), (d, f), (e, f)$

Speicheraufwand: $O(m \log(n))$.

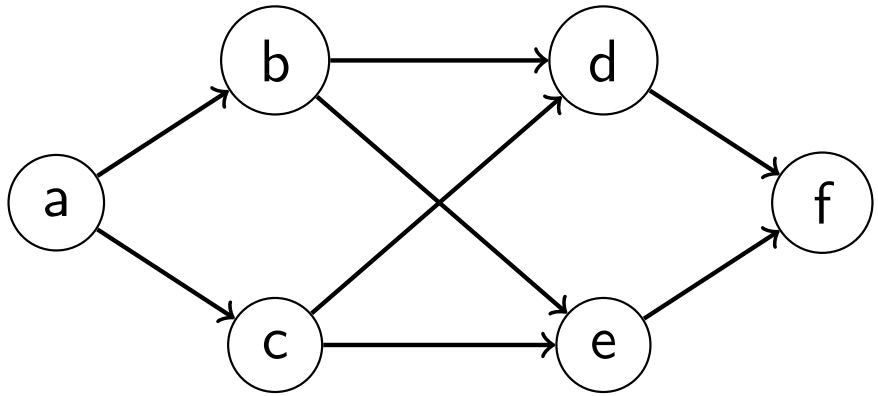
Adjazenzliste



a	b,c
b	a,d,e
c	a,d,e
d	b,c,f
e	b,c,f
f	d,e

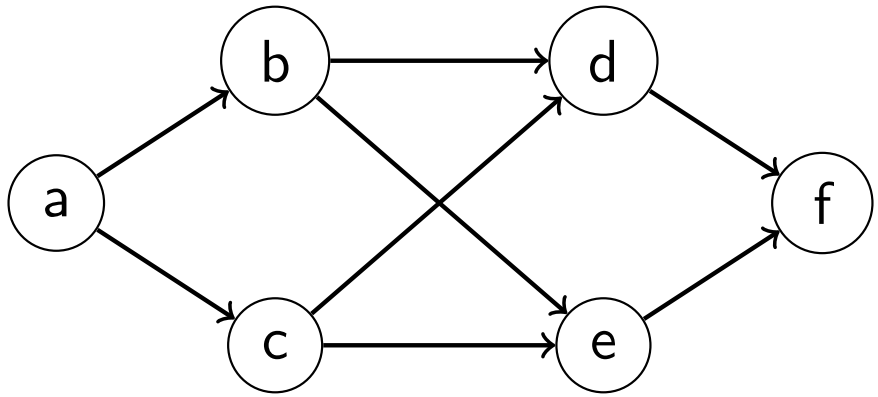
Speicheraufwand: $O(m \log n + n \log m)$.

Inzidenzmatrix von Digraphen



	ab	ac	bd	be	cd	ce	df	ef
a	1	1						
b	-1		1	1				
c		-1			1	1		
d			-1		-1		1	
e				-1		-1		1
f							-1	-1

Adjazenzmatrix von Digraphen



	a	b	c	d	e	f
a		1	1			
b				1	1	
c			1	1		
d						1
e						1
f						

Speicheraufwand: $O(n^2)$.

Graph Scanning Algorithmus

Input: Ein Graph G/ Digraph D und ein Knoten r von G/D.

Output: Die Menge R der Knoten, die in G/D von r aus auf Wegen/gerichteten Wegen erreichbar sind und eine Menge $T \subseteq E(G)/A(D)$, für die (R, T) Baum/Arboreszenz mit Wurzel r ist.

begin

  $R \leftarrow \{r\}$; $Q \leftarrow \{r\}$; $T \leftarrow \emptyset$; $l(r) \leftarrow 0$;

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

1 Wähle $v \in Q$;

2 **if** $\exists\ w \in V \setminus R$ *mit* $e = vw \in E(G)$ / $e = (v, w) \in A(D)$ **then**

      Wähle solch ein w;

      $R \leftarrow R \cup \{w\}$; $Q \leftarrow Q \cup \{w\}$; $T \leftarrow T \cup \{e\}$; $l(w) \leftarrow l(v) + 1$;

    else

      $Q \leftarrow Q \setminus \{v\}$;

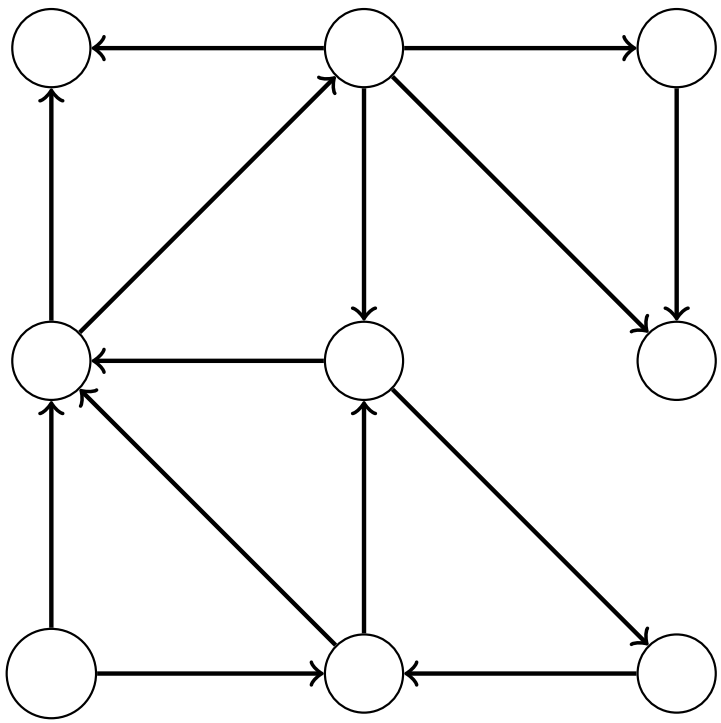
    end

  end

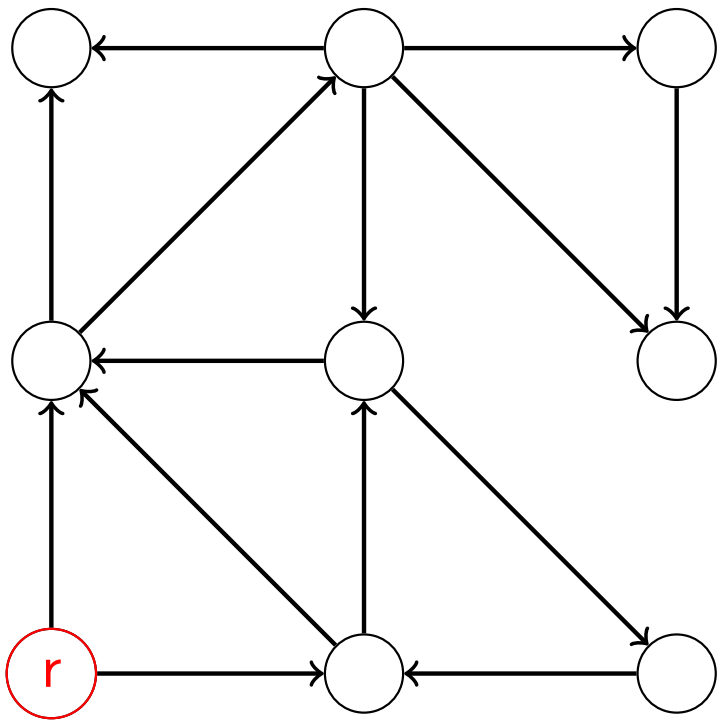
  return (R, T);

end

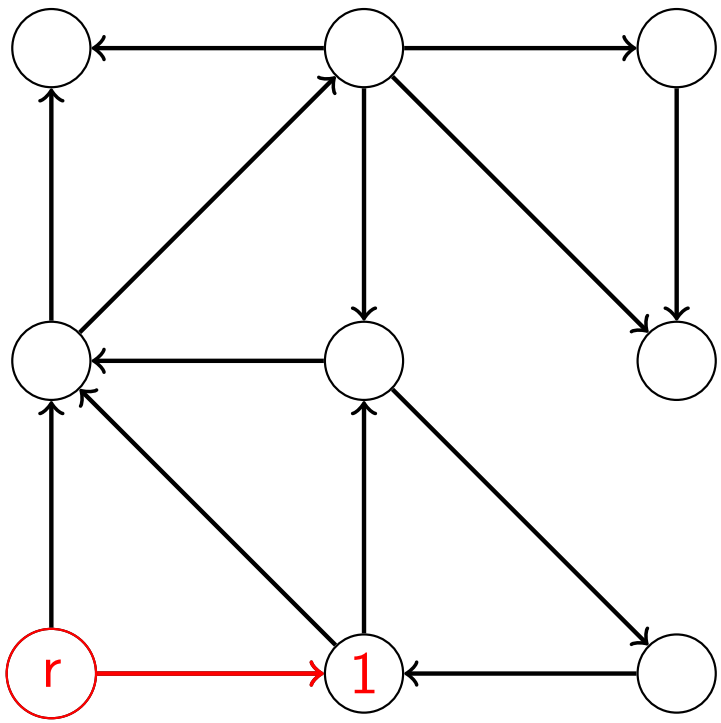
Graph Scanning Algorithmus



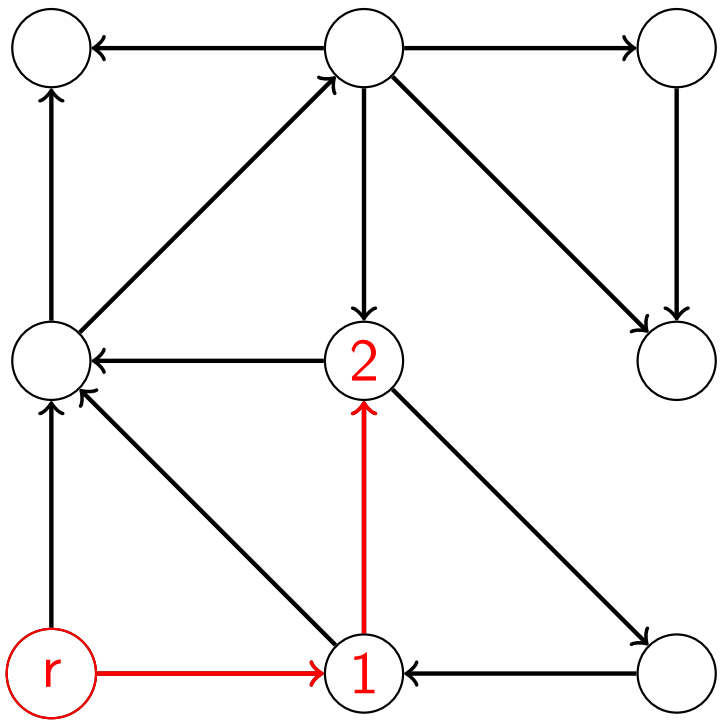
Graph Scanning Algorithmus



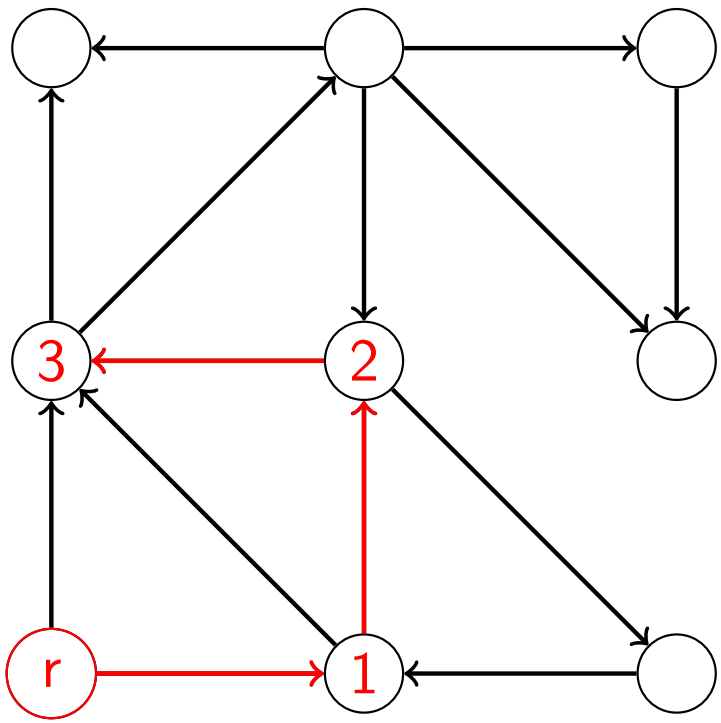
Graph Scanning Algorithmus



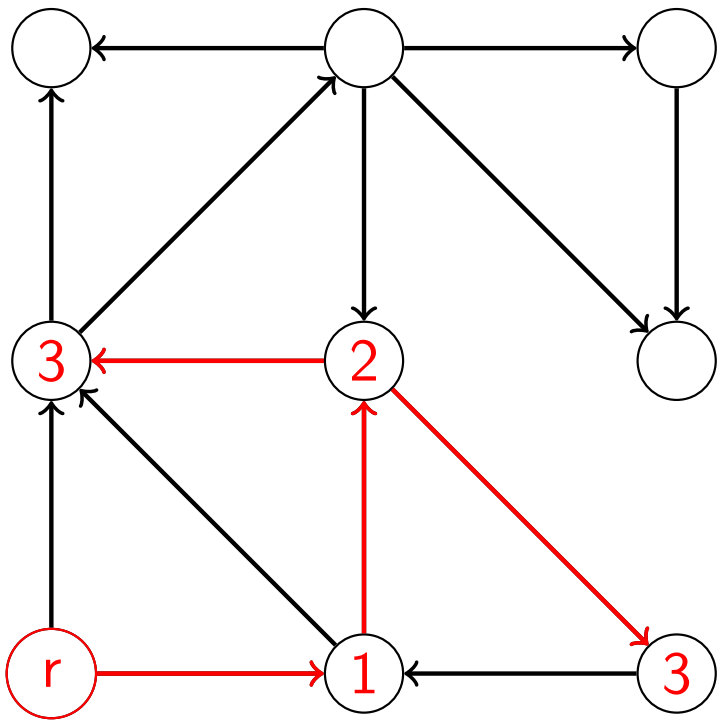
Graph Scanning Algorithmus



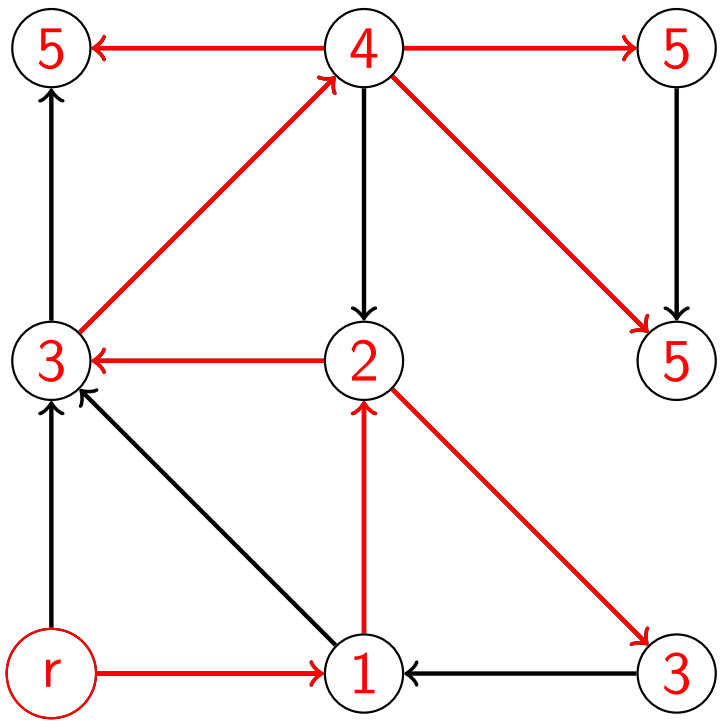
Graph Scanning Algorithmus



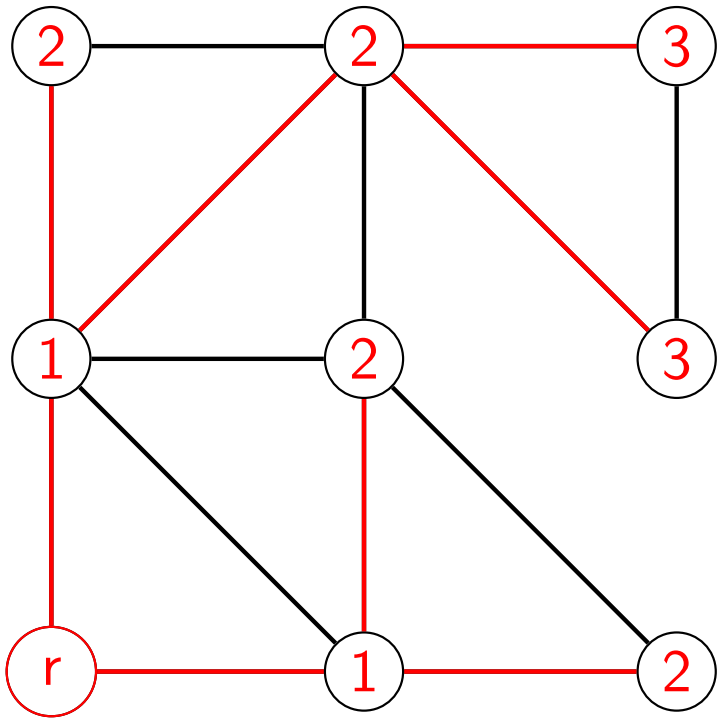
Graph Scanning Algorithmus



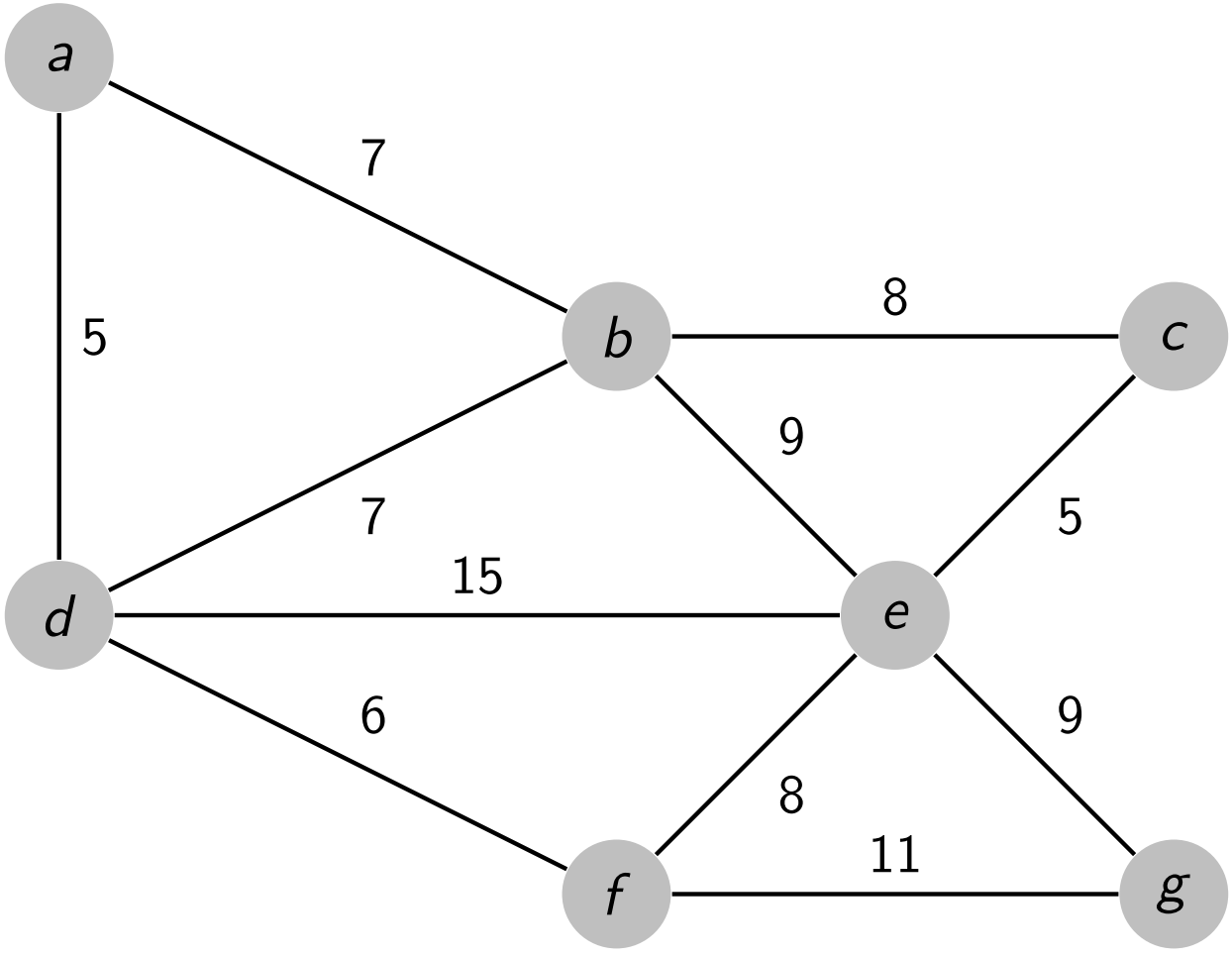
Graph Scanning Algorithmus



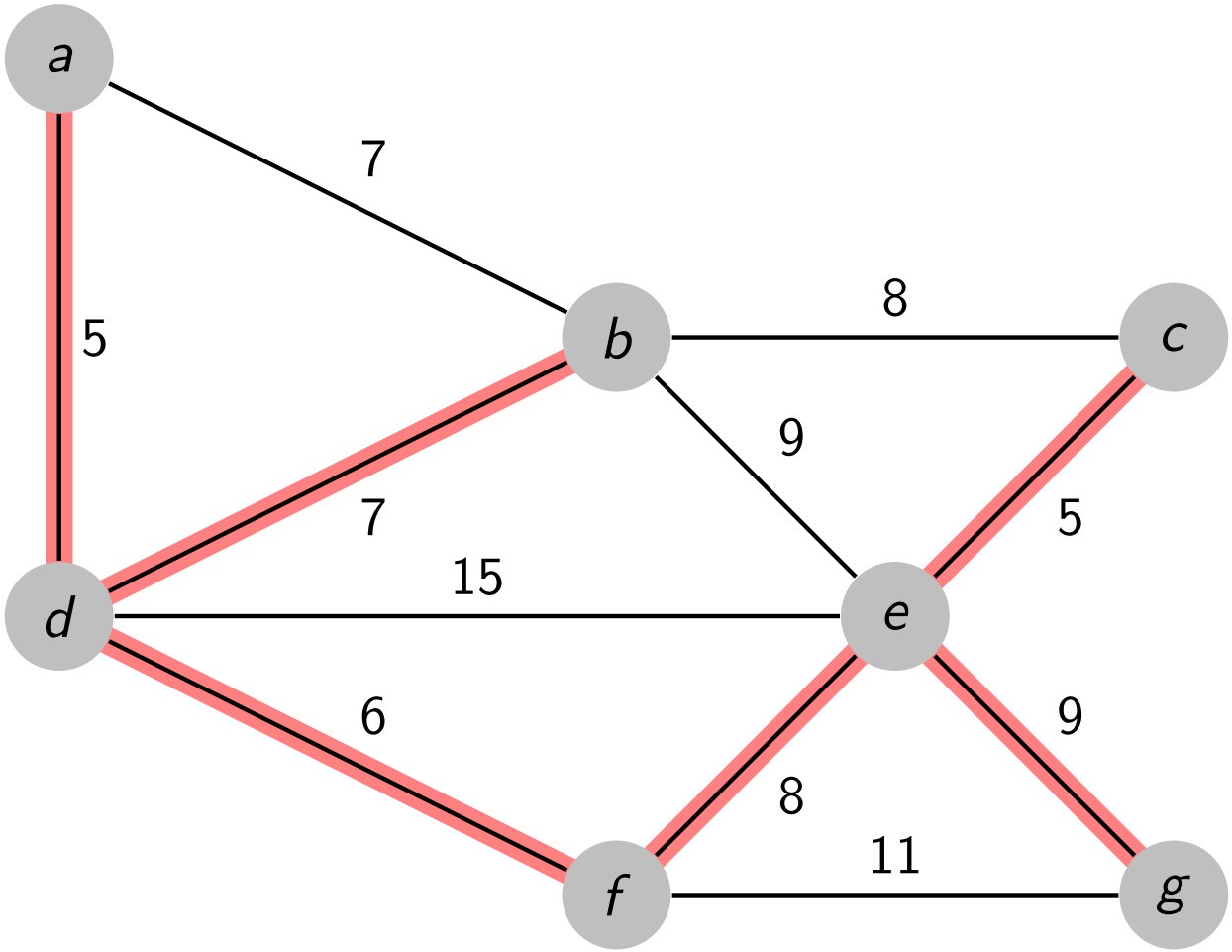
BFS-Bäume



Minimum Spanning Trees



Minimum Spanning Trees



Kruskal's Algorithm

Input: Ein zusammenhängender Graph G und eine Kostenfunktion $c : E \to \mathbb{R}$.

Output: Ein Minimum Spanning Tree T f"ur (G, c).

begin

  Sortiere die Kanten von G so, dass $c(e_1) \leq c(e_2) \leq ... \leq c(e_m)$ gilt;

  $T \leftarrow (V(G), \emptyset)$;

  for $i = 1$ **to** m **do**

    if $T + e_i$ *ist ein Wald* **then**

      $T \leftarrow (V(G), E(T) \cup \{e_i\})$;

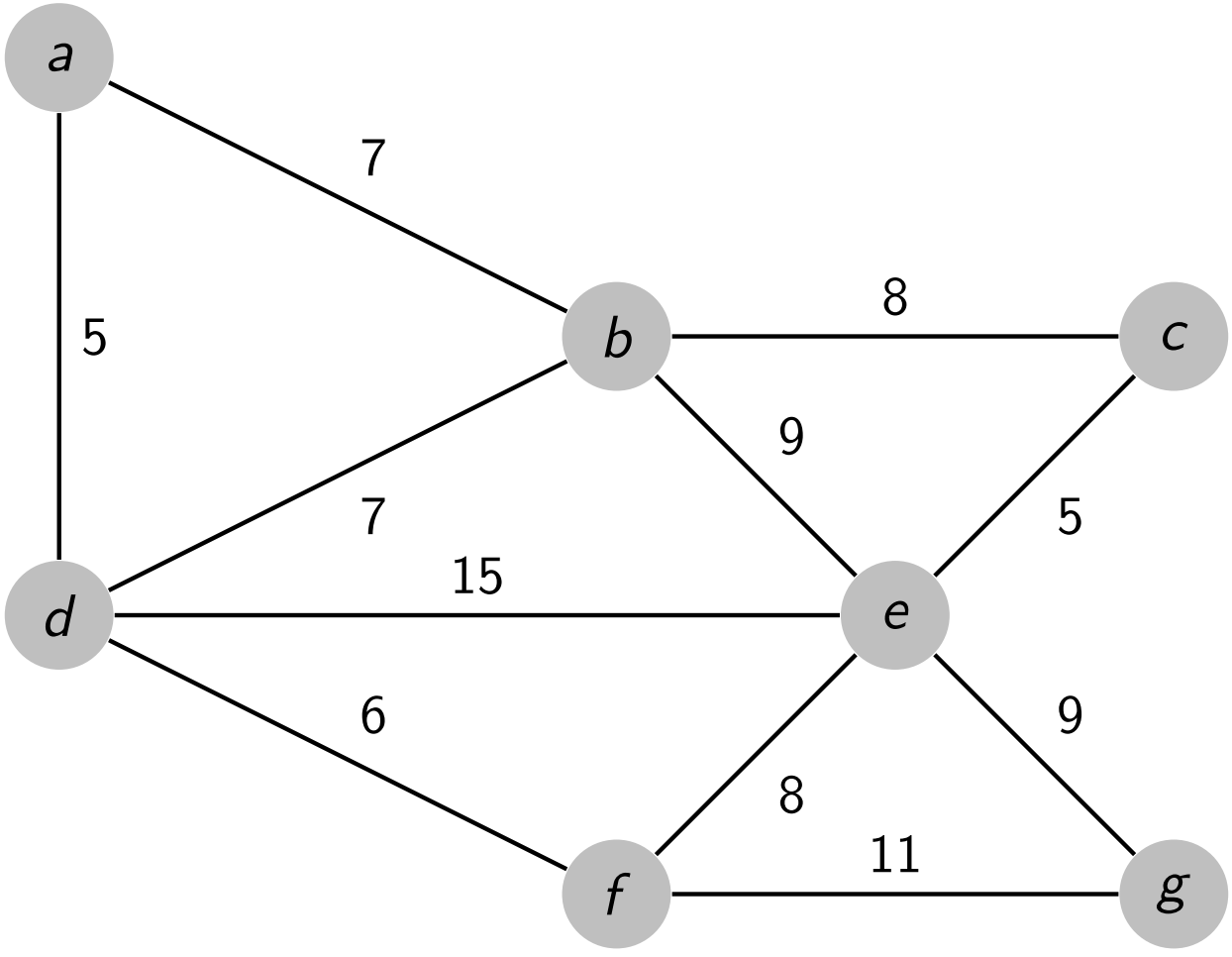
    end

  end

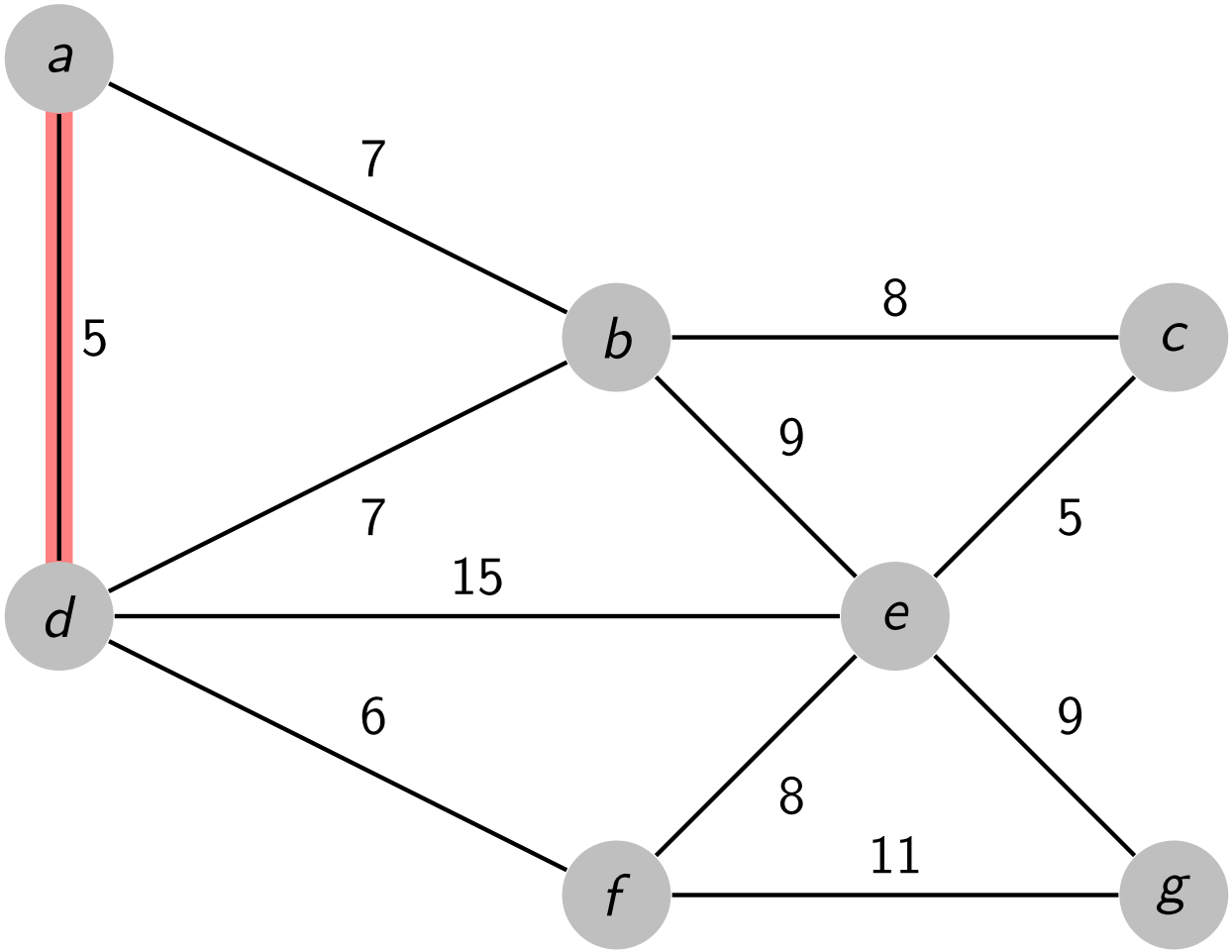
  return T;

end

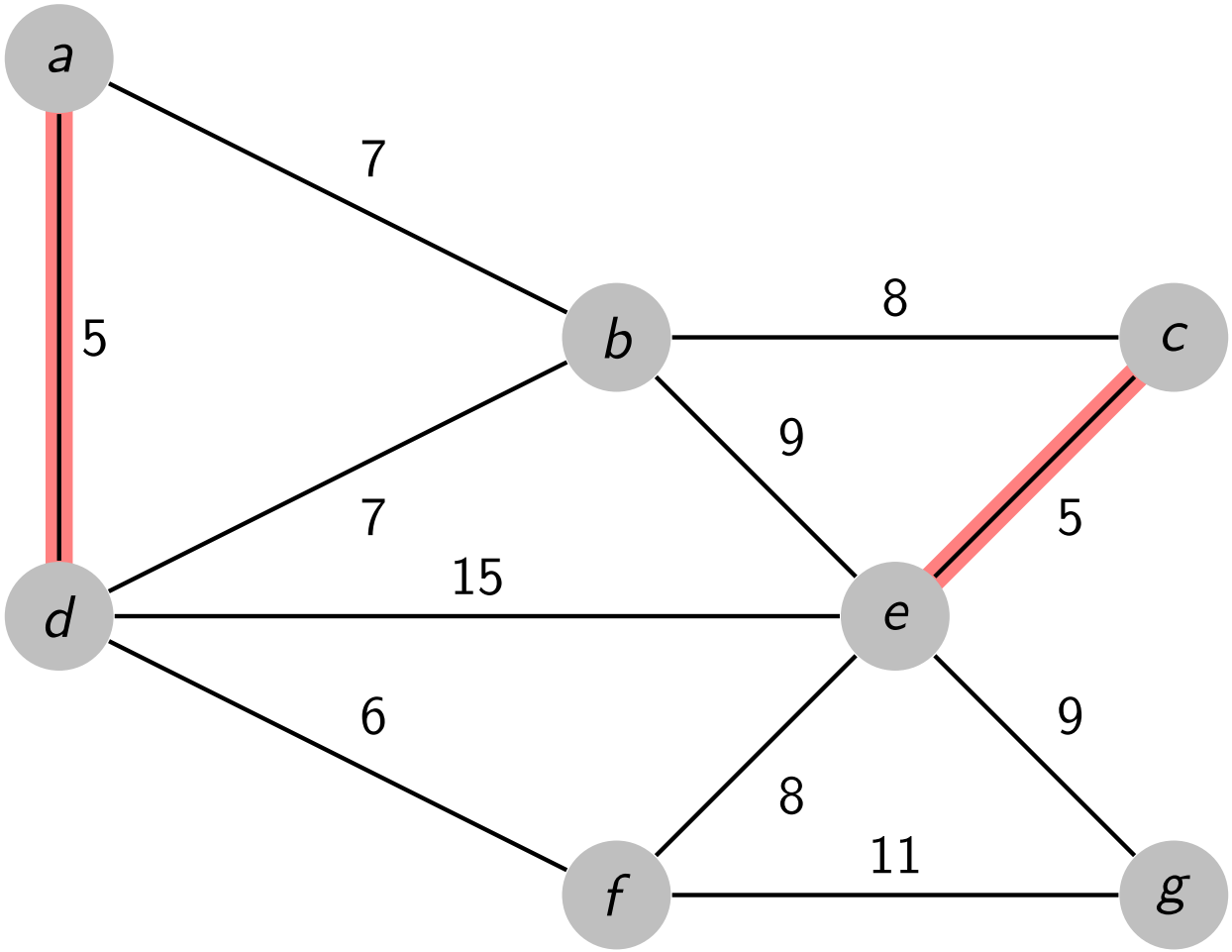
Kruskals's algorithm



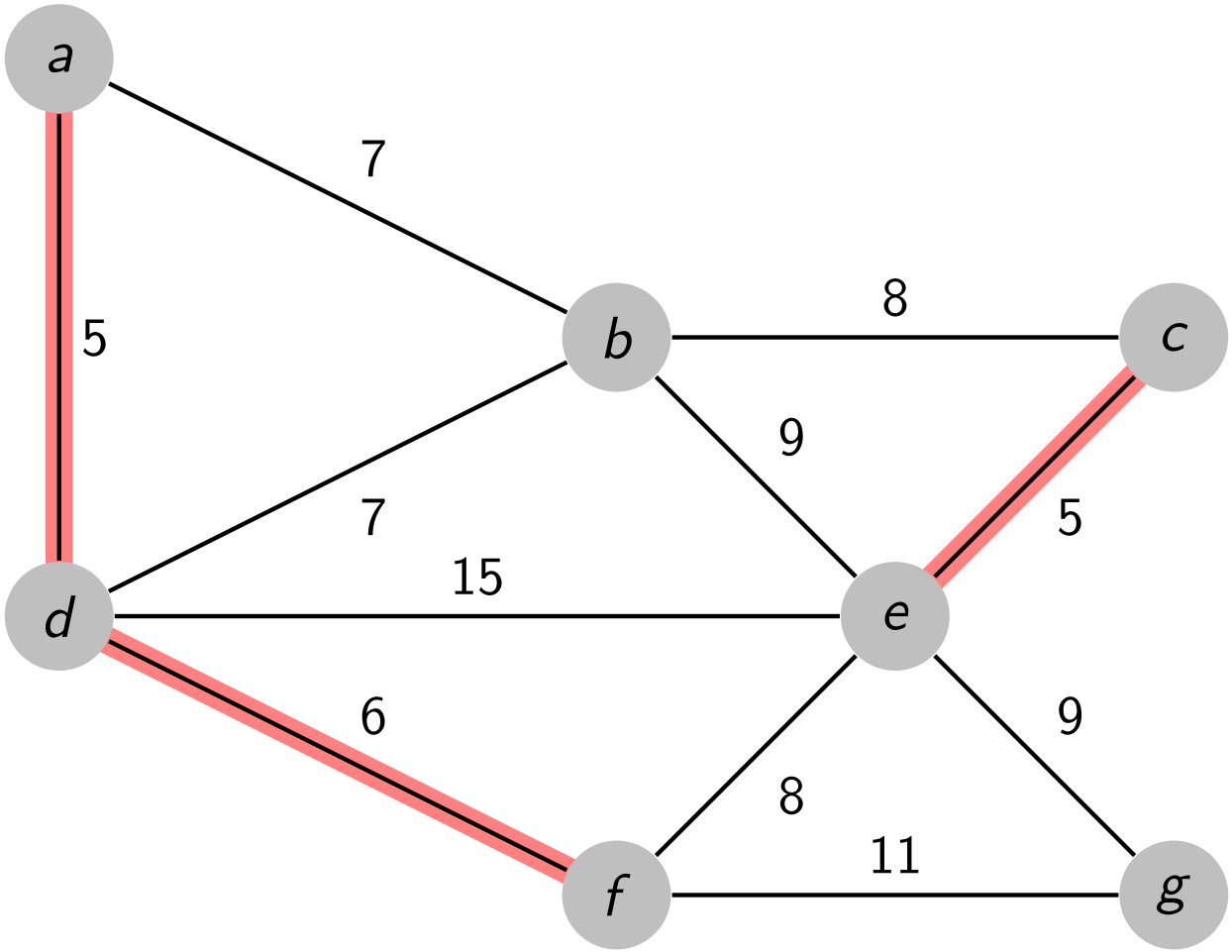
Kruskals's algorithm



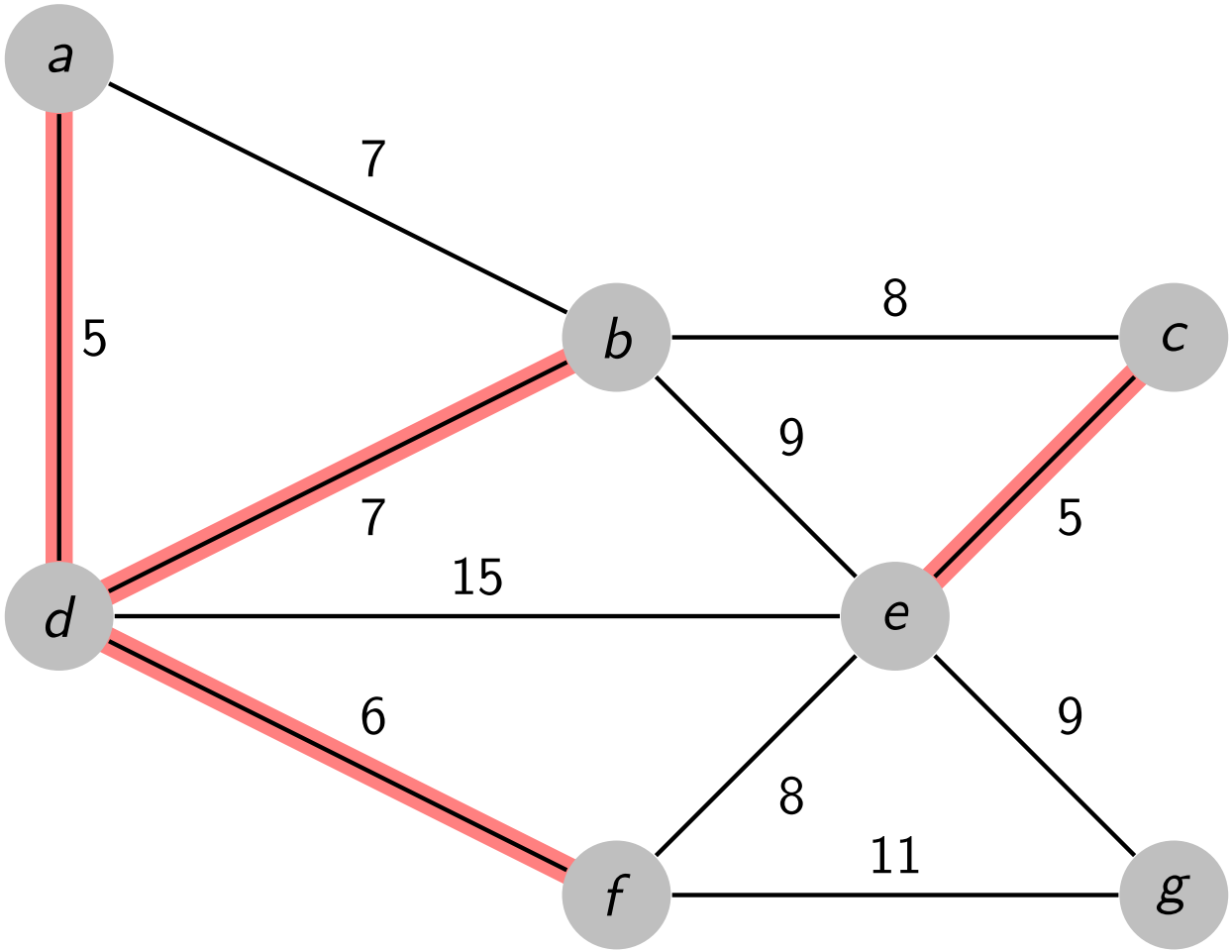
Kruskals's algorithm



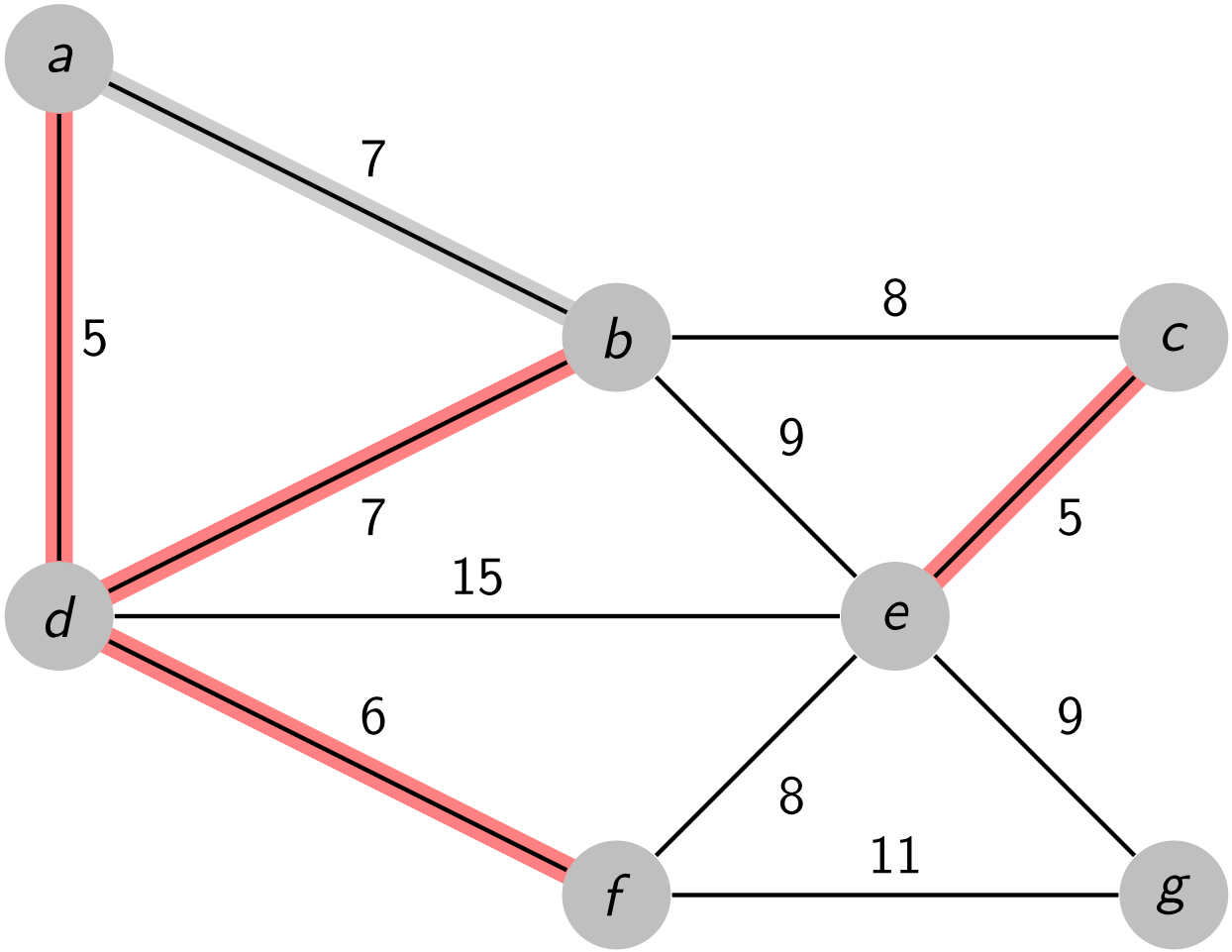
Kruskals's algorithm



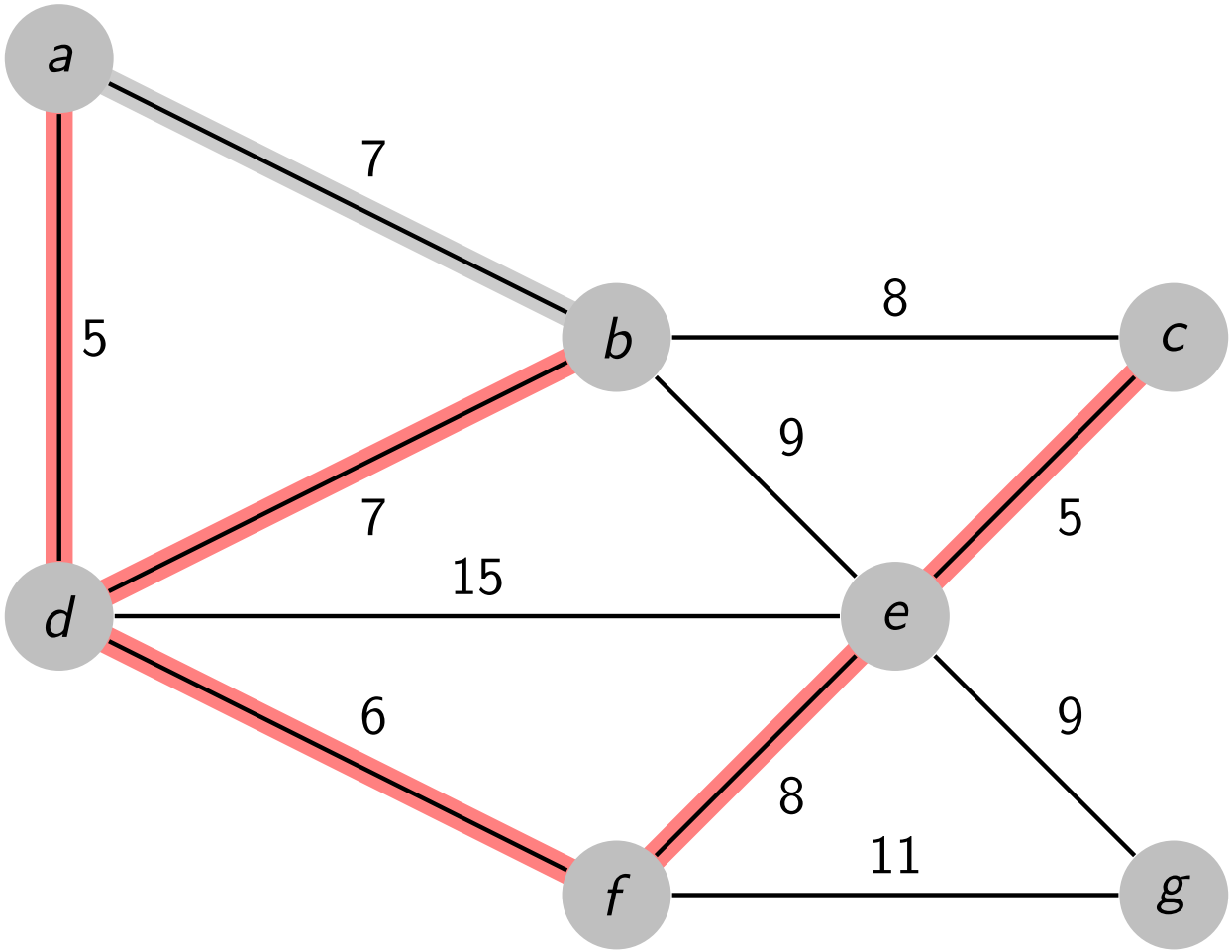
Kruskals's algorithm



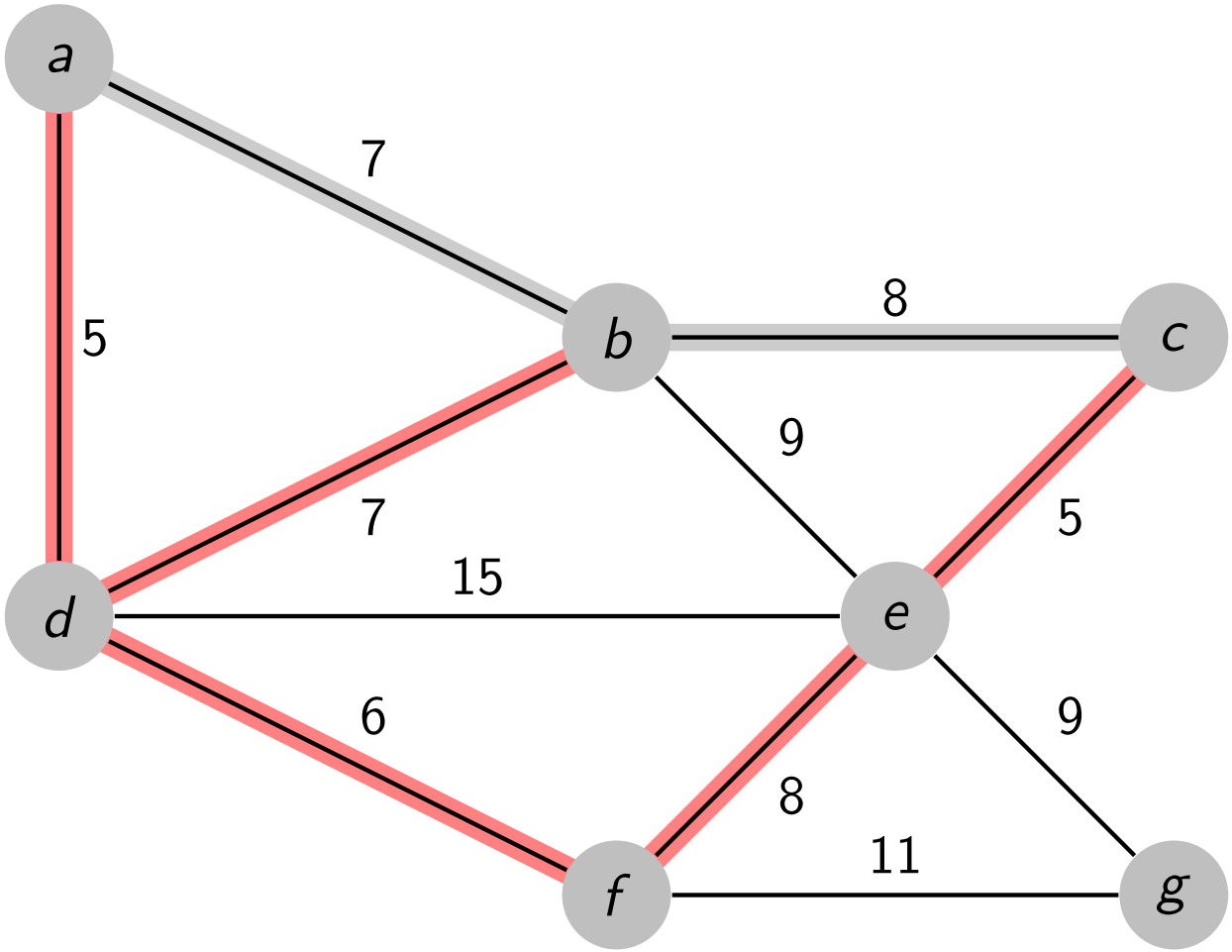
Kruskals's algorithm



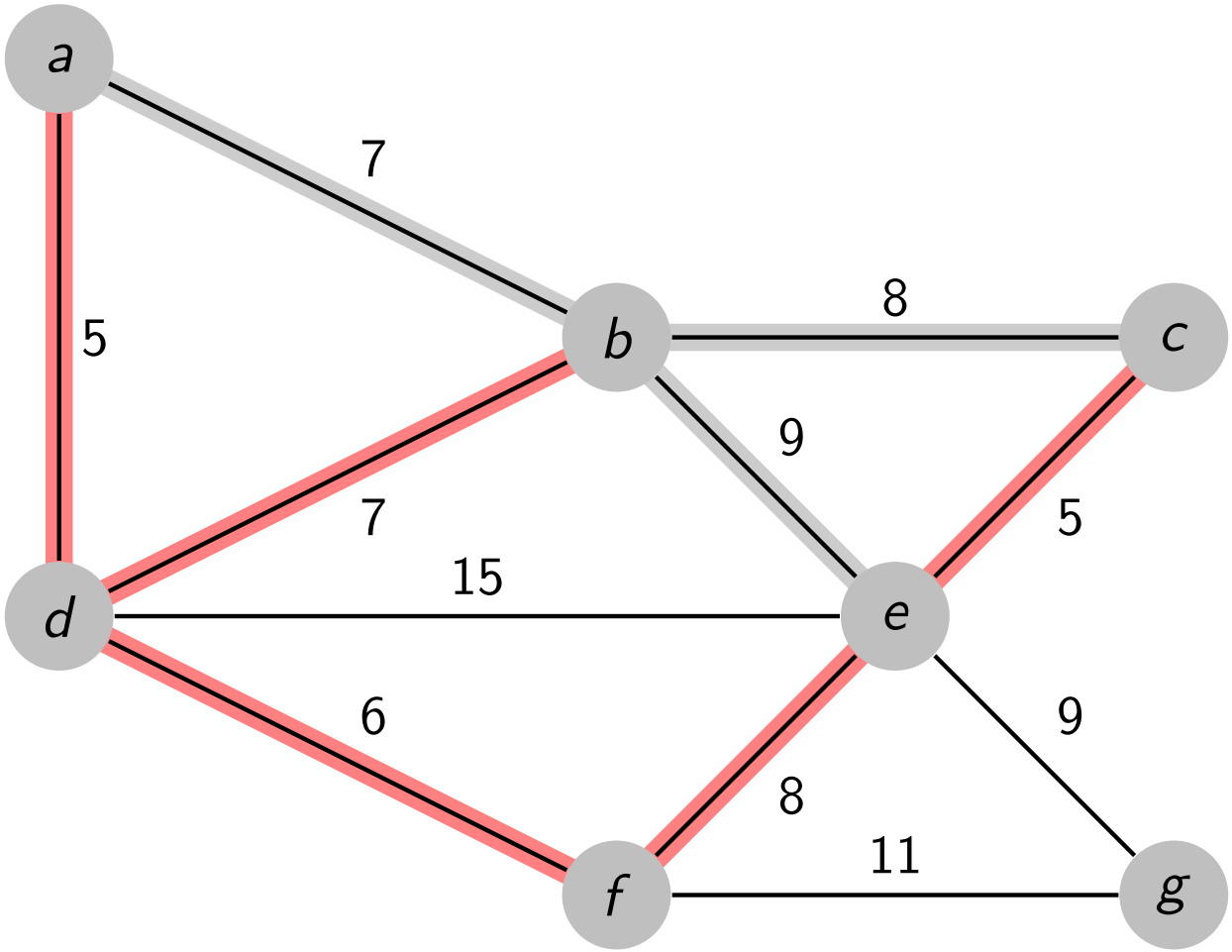
Kruskals's algorithm



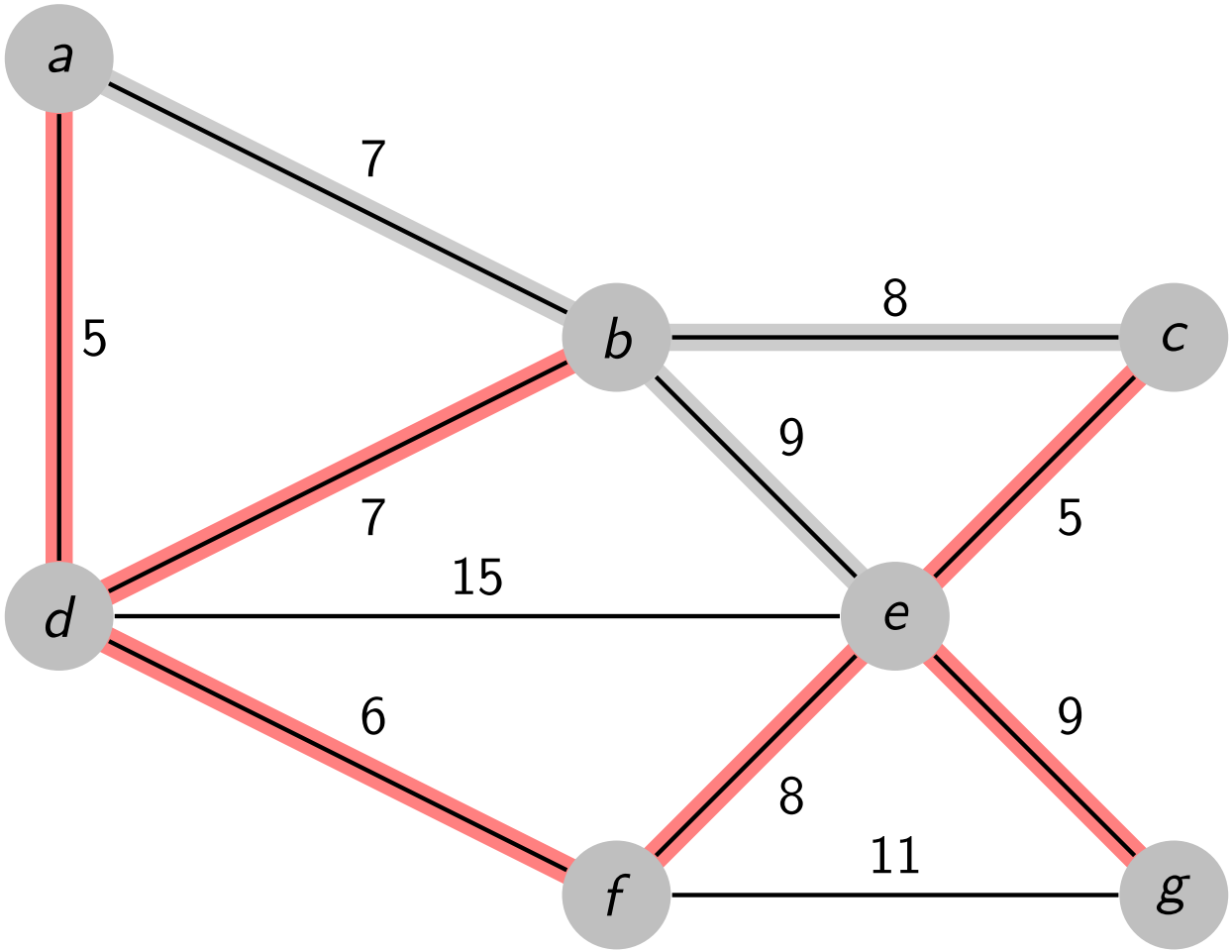
Kruskals's algorithm



Kruskals's algorithm



Kruskals's algorithm



Prim's Algorithm

Input: Ein zusammenhängender Graph G und eine Kostenfunktion $c : E(G) \to \mathbb{R}$.

Output: Ein Minimum Spanning Tree T.

begin

  W"ahle $v \in V(G)$;

  $T \leftarrow (\{v\}, \emptyset)$;

  while $V(T) \neq V(G)$ **do**

    Wähle eine Kante xy minimaler Kosten $c(xy)$ mit $x \in V(T)$ und $y \in V(G) \setminus V(T)$;

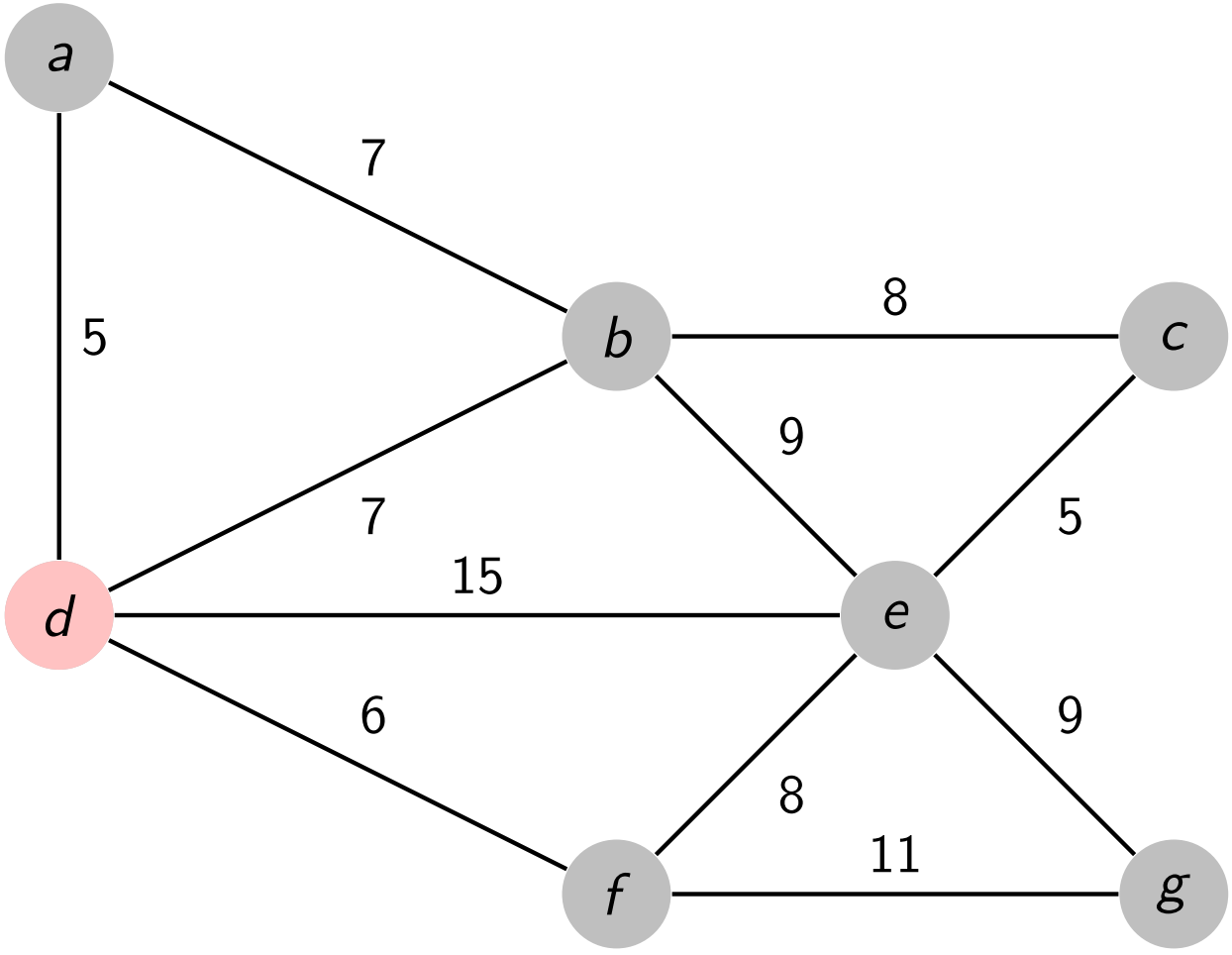
    $T \leftarrow (V(T) \cup \{y\}, E(T) \cup \{e\})$;

  end

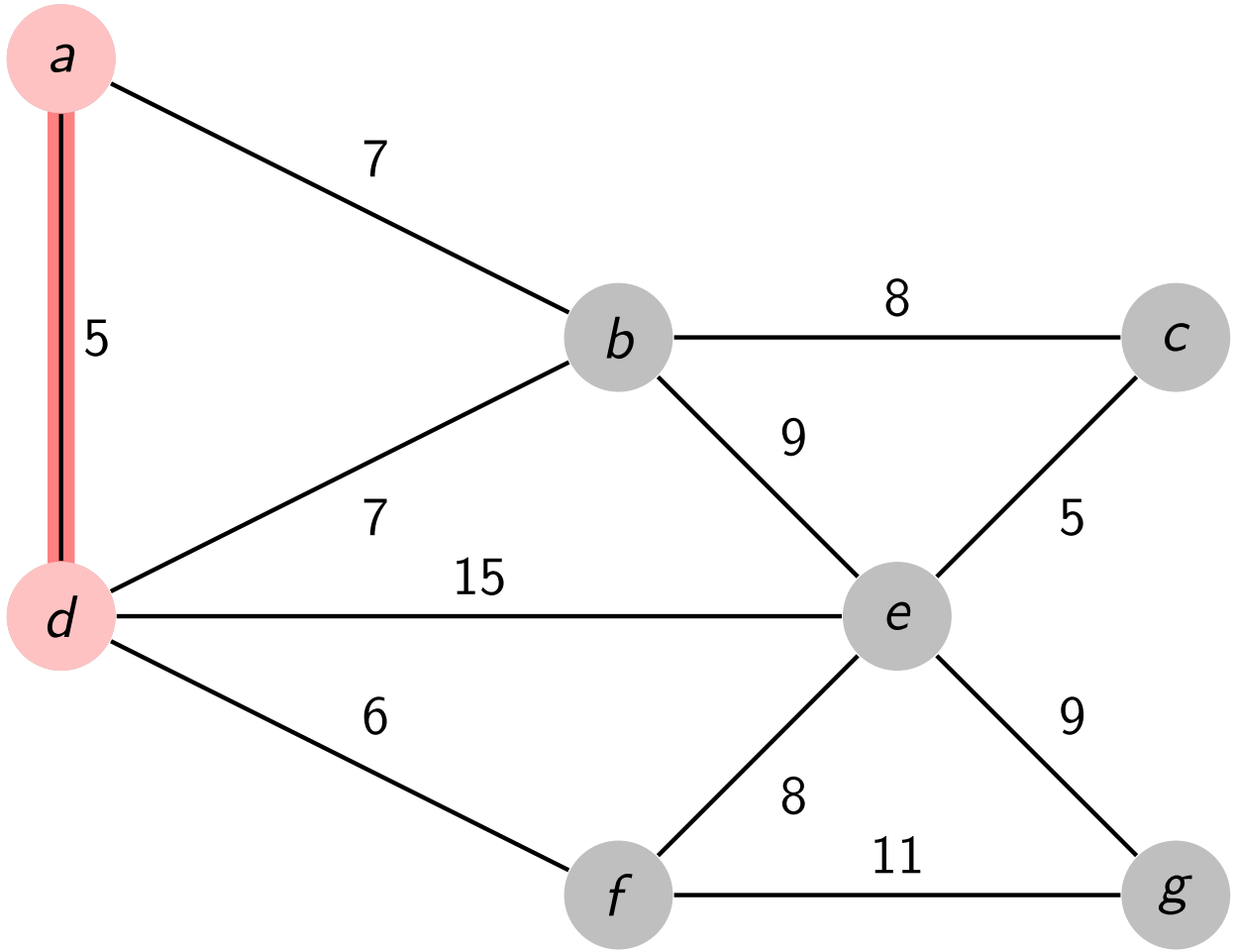
  return T;

end

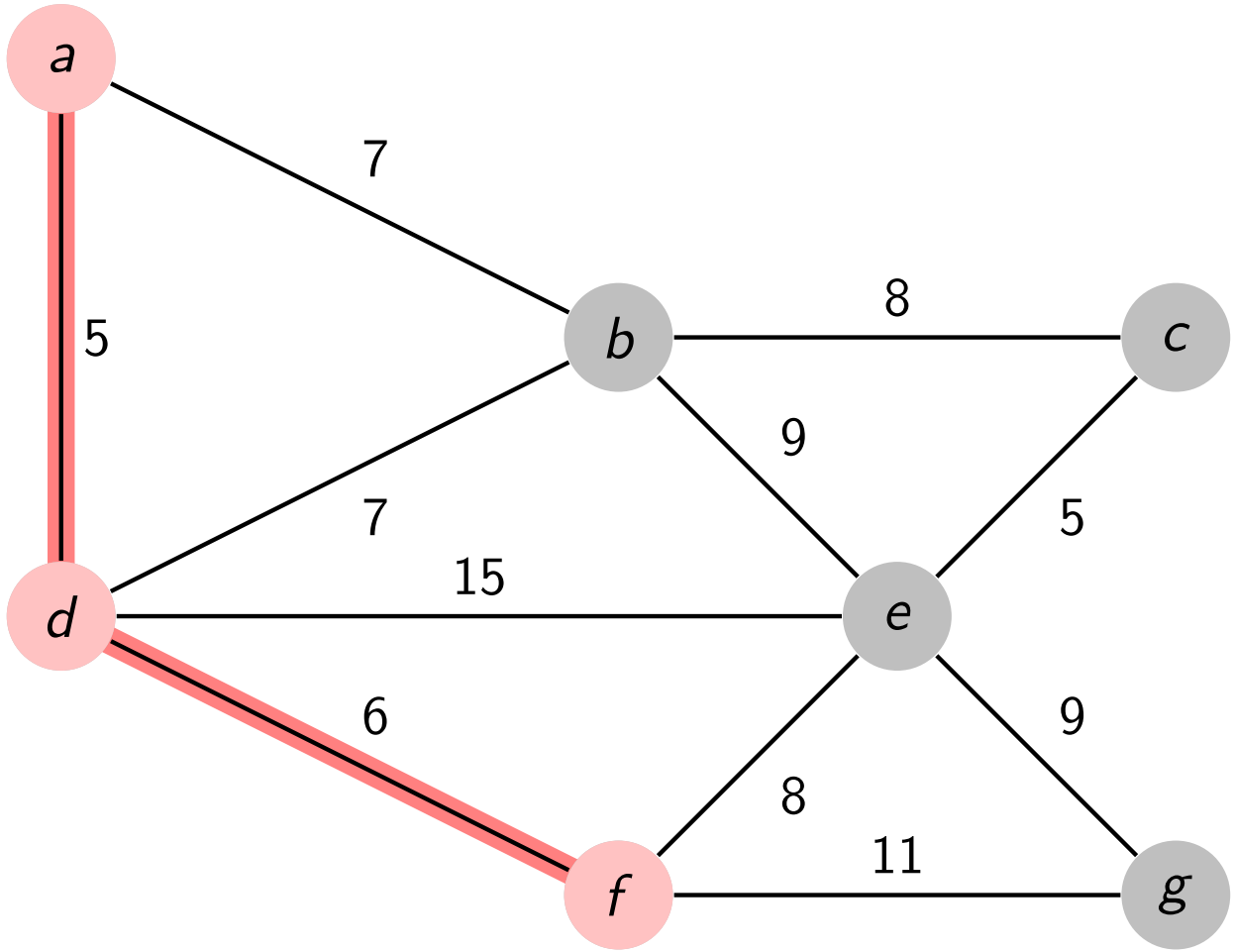
Prim's algorithm



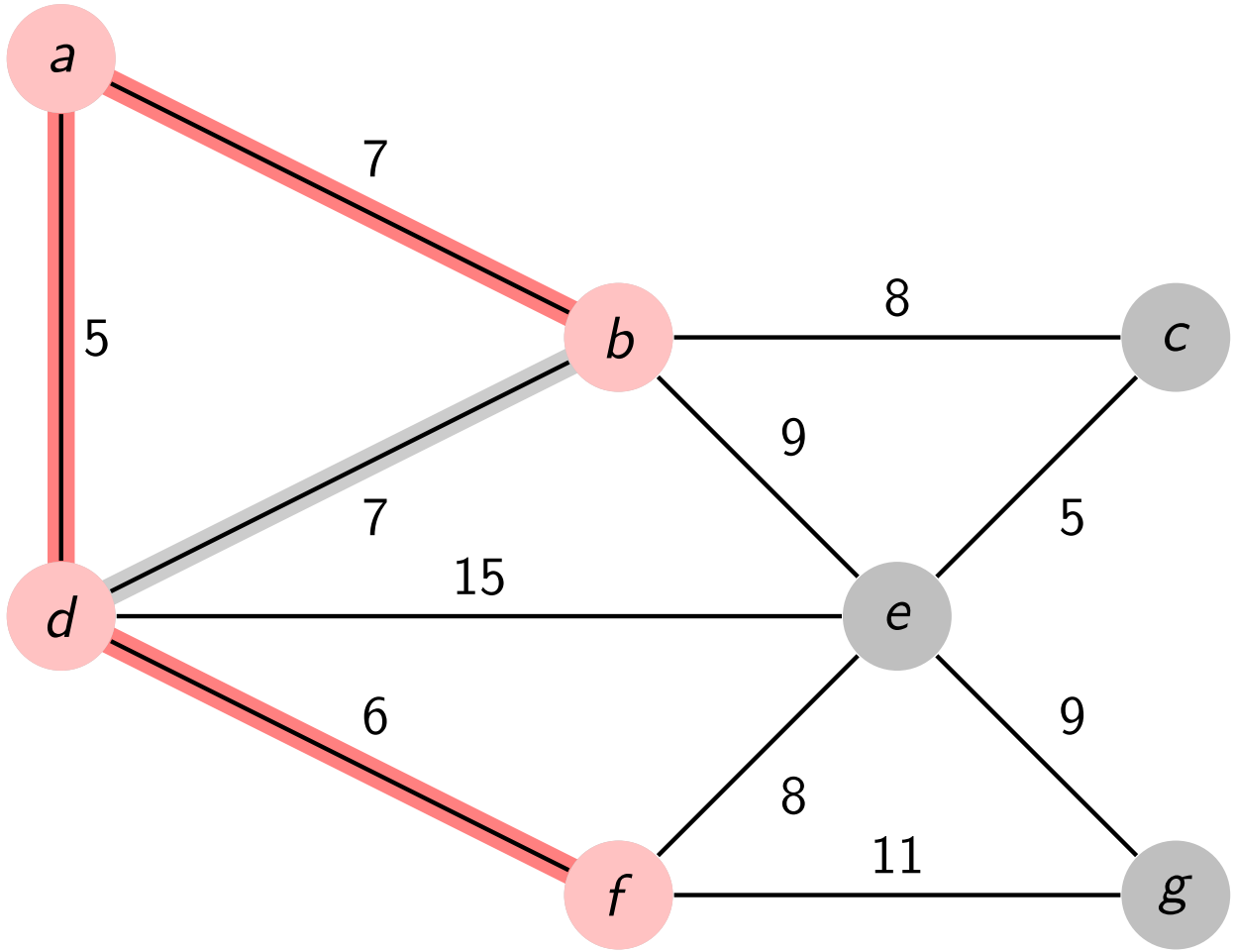
Prim's algorithm



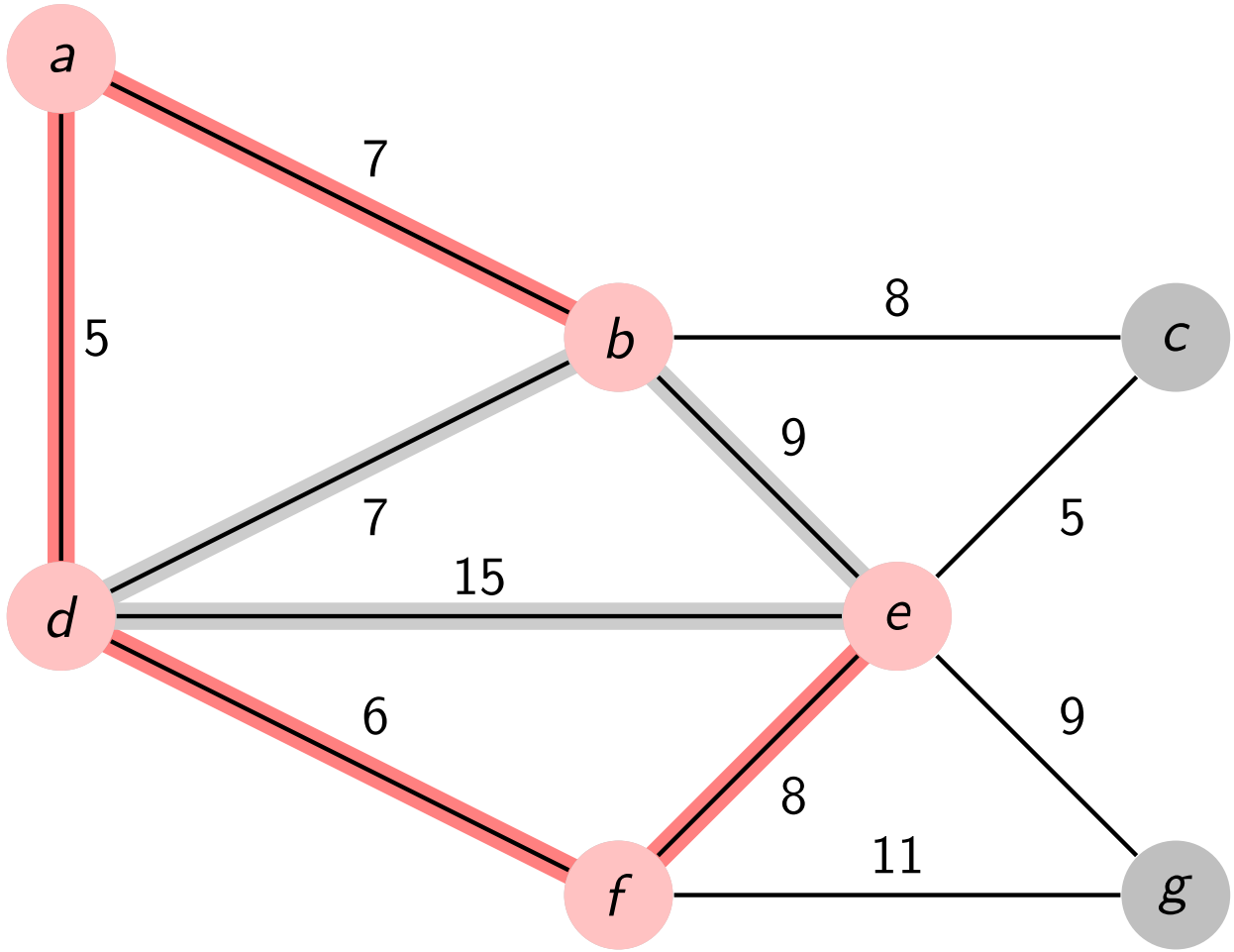
Prim's algorithm



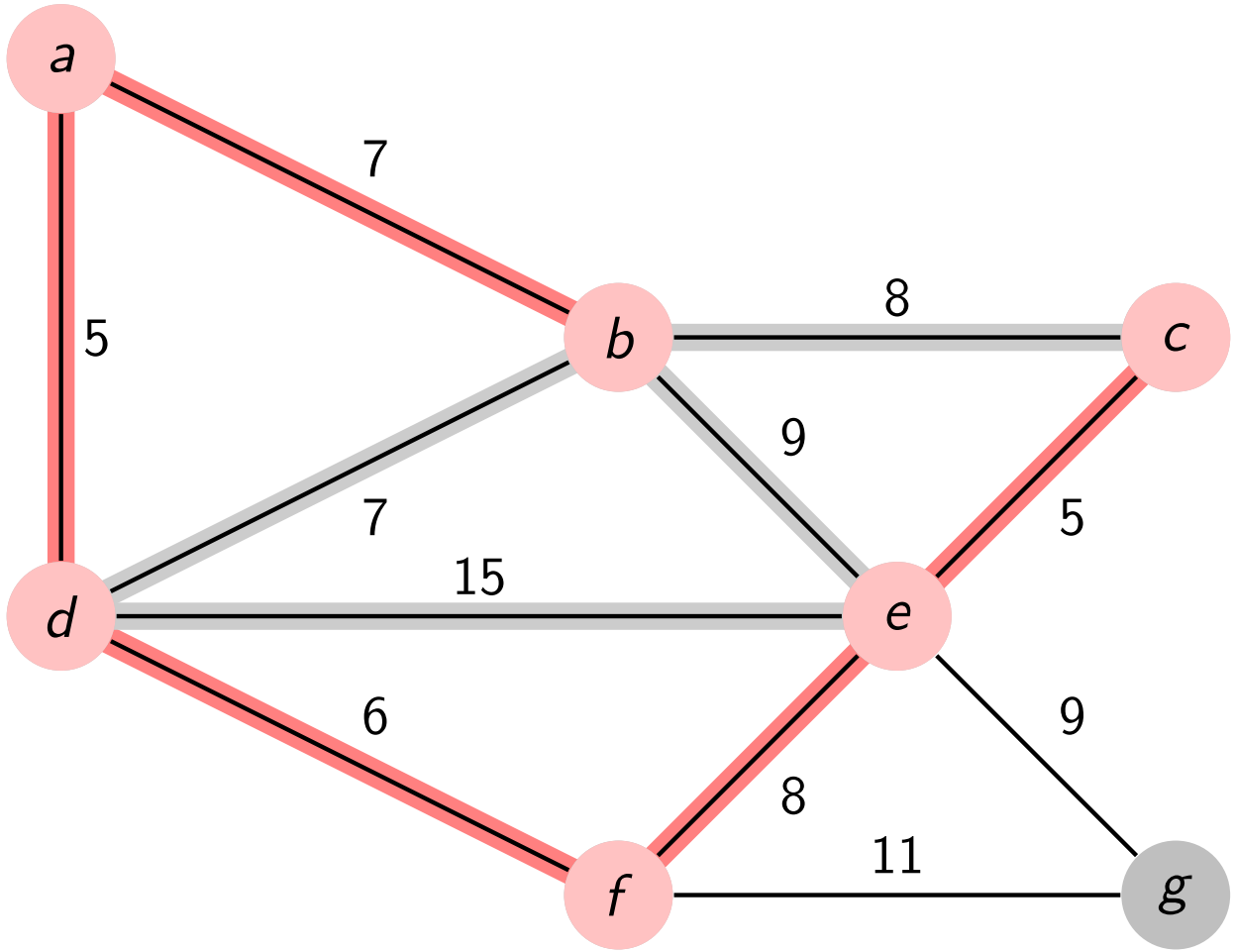
Prim's algorithm



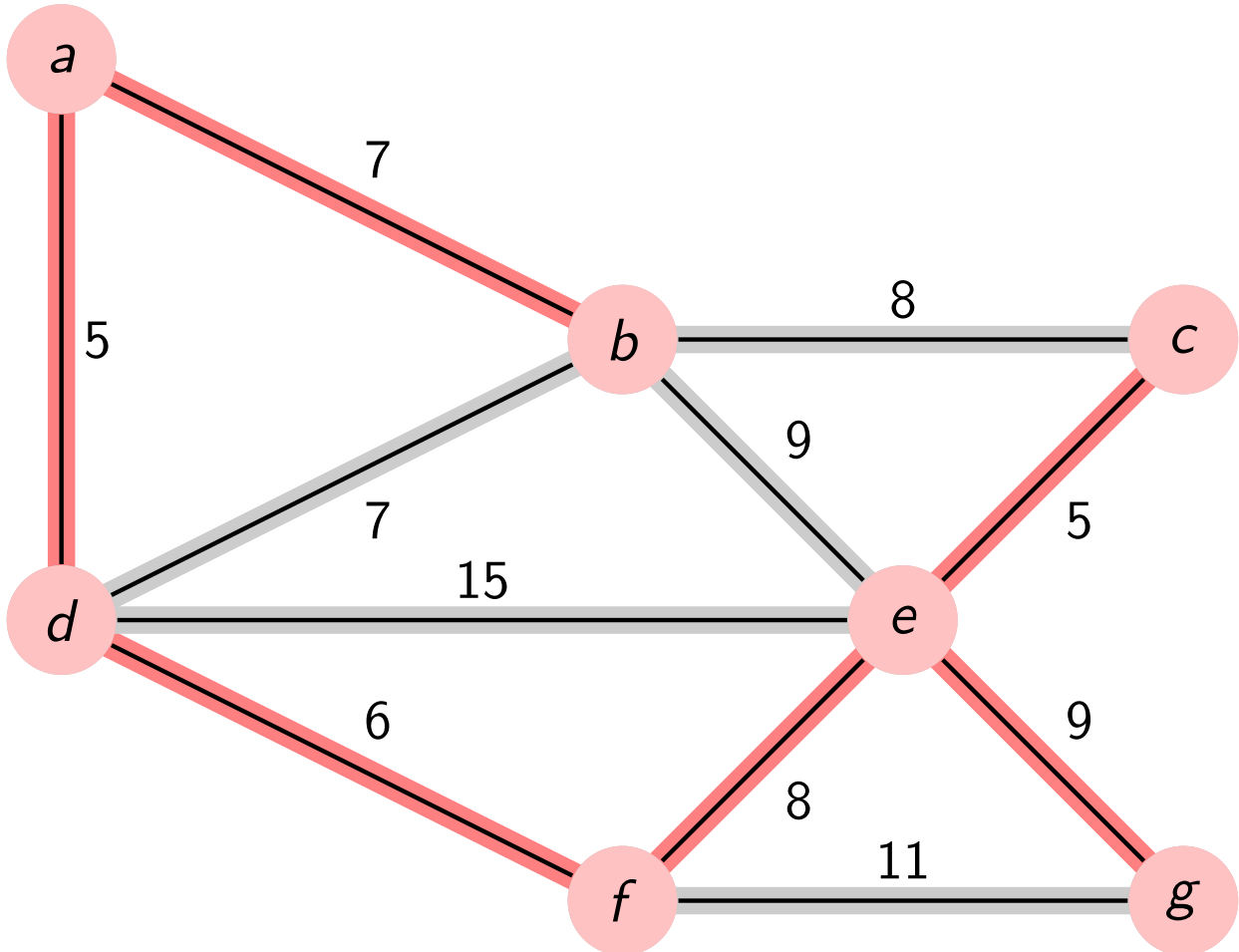
Prim's algorithm



Prim's algorithm



Prim's algorithm



Dijkstra's algorithm

Input: Ein Digraph D, $c : A(D) \to \mathbb{R}_{>0}$ und $r \in V(D)$.

Output: Kürzeste Wege von r zu allen erreichbaren Knoten: $(l(v), p(v))_{v \in V(D) \setminus \{r\}}$, wobei $l(v) = \mathrm{dist}_{(D,c)}(r, v)$ gilt und $p(v)$ ein Vorgänger von v auf einem bzgl. c kürzesten Weg von r nach v ist. Ist v nicht erreichbar, so ist $l(v) = \infty$ und $p(v)$ nicht definiert.

begin

  $l(r) \leftarrow 0$;

  for $v \in V(D) \setminus \{r\}$ **do** $l(v) \leftarrow \infty$;

1 $R \leftarrow \emptyset$;

  while $R \neq V(D)$ **do**

2 Wähle $v \in V(D) \setminus R$ mit $l(v) = \min\{l(w) \mid w \in V(D) \setminus R\}$;

3 $R \leftarrow R \cup \{v\}$;

  for $w \in V(D) \setminus R$ *mit* $(v, w) \in A(D)$ **do**

    if $l(w) > l(v) + c((v, w))$ **then**

4 $l(w) \leftarrow l(v) + c((v, w))$; $p(w) \leftarrow v$;

    end

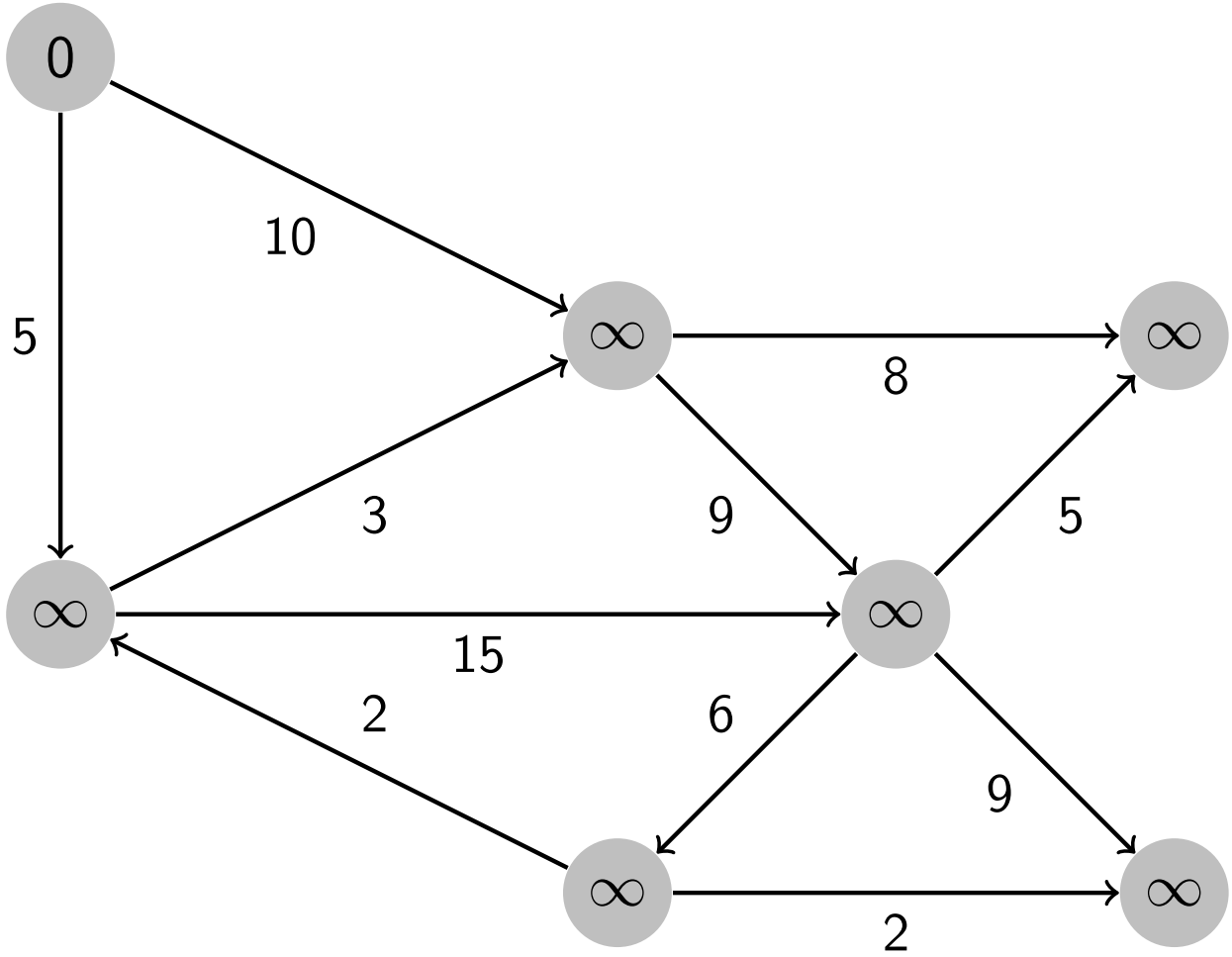
  end

  end

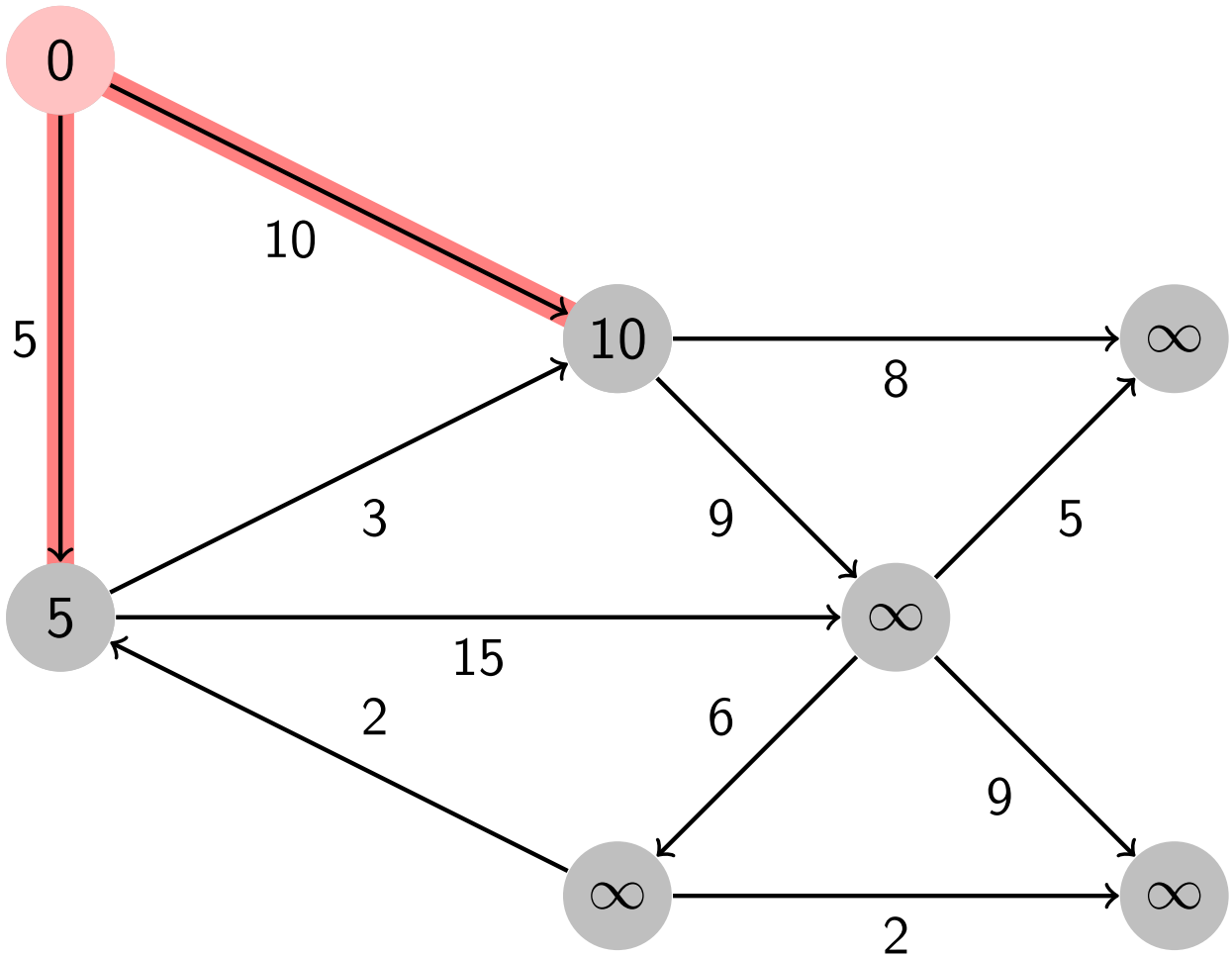
  return $(l(v), p(v))_{v \in V(D) \setminus \{r\}}$;

end

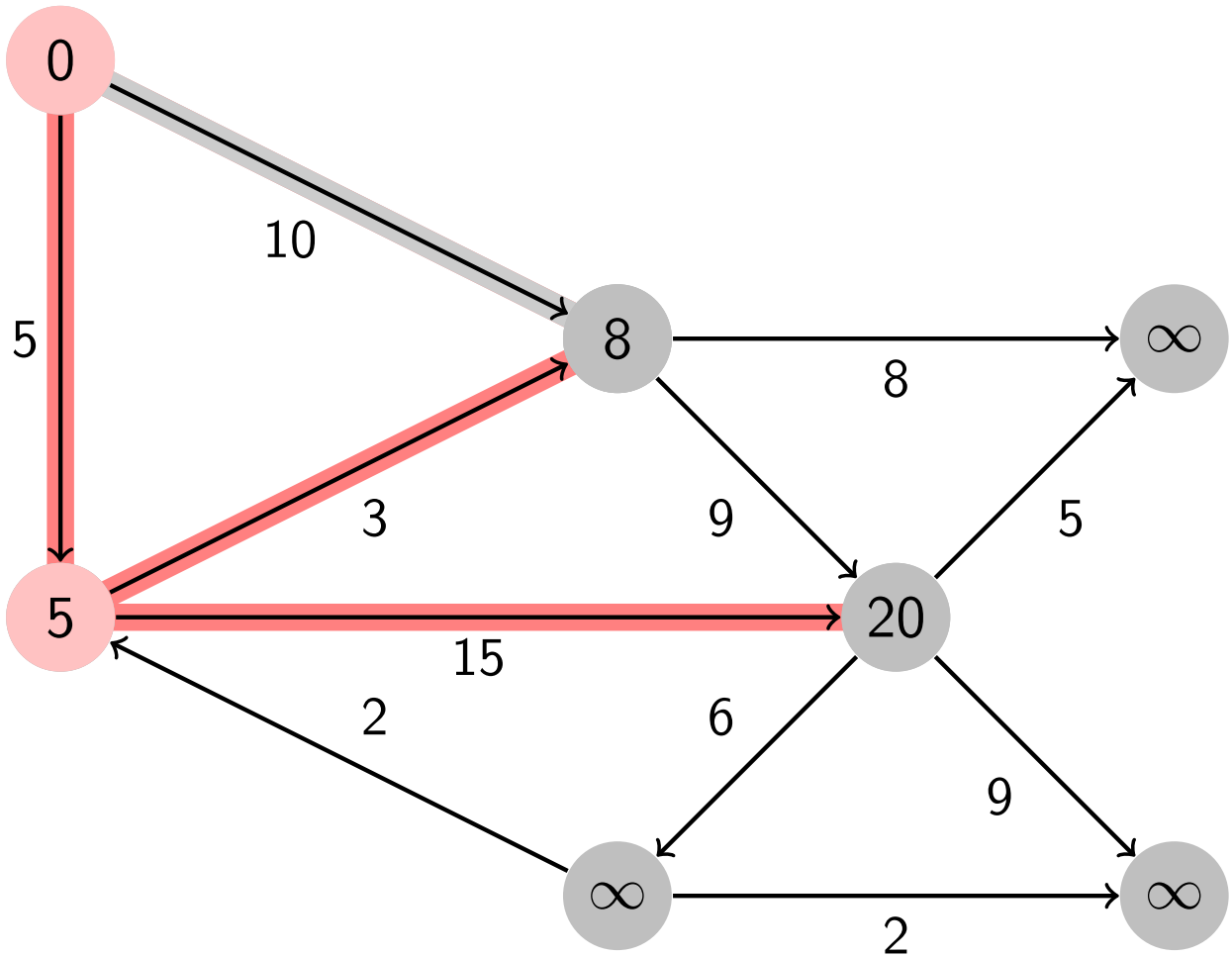
Dijkstra's algorithm



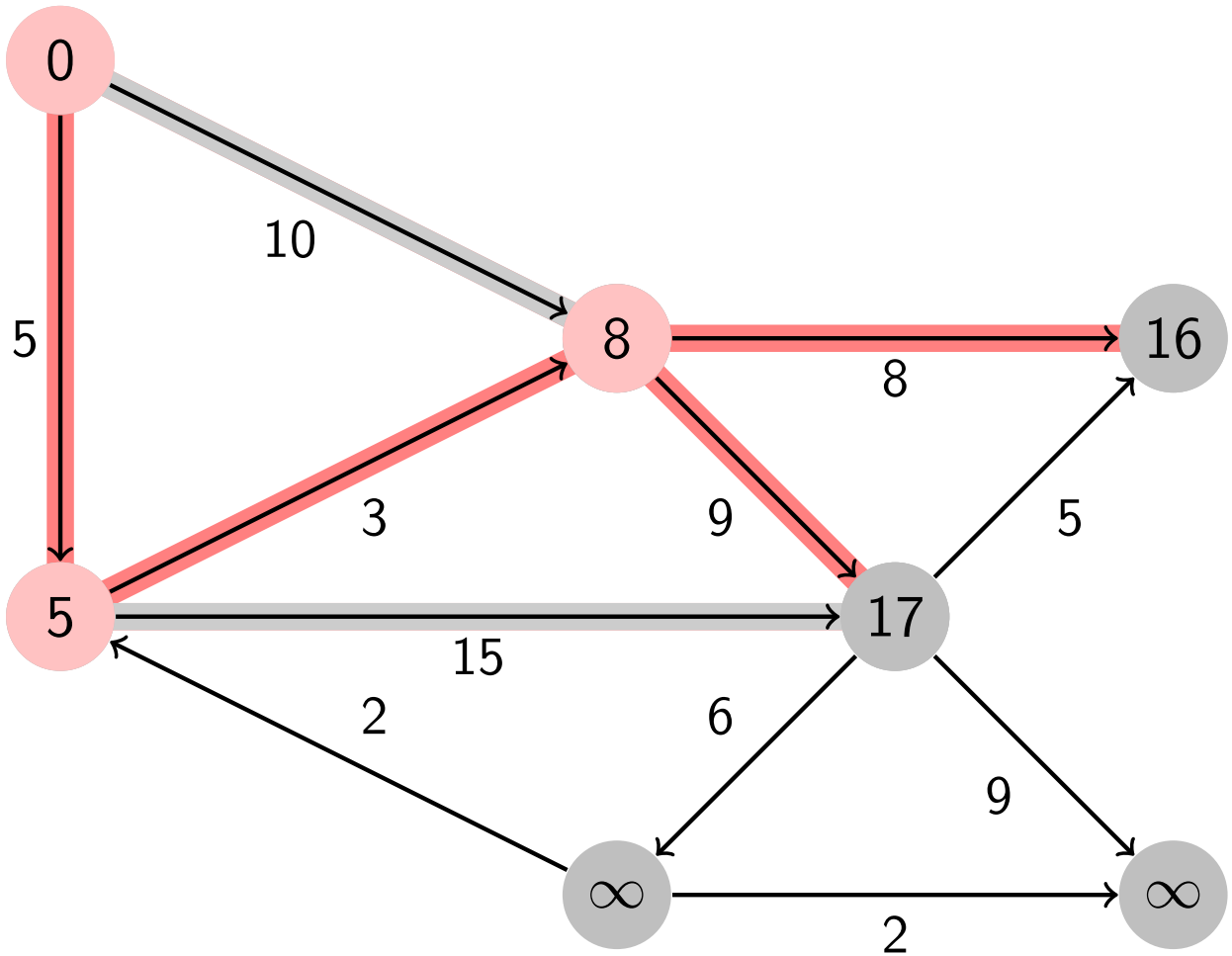
Dijkstra's algorithm



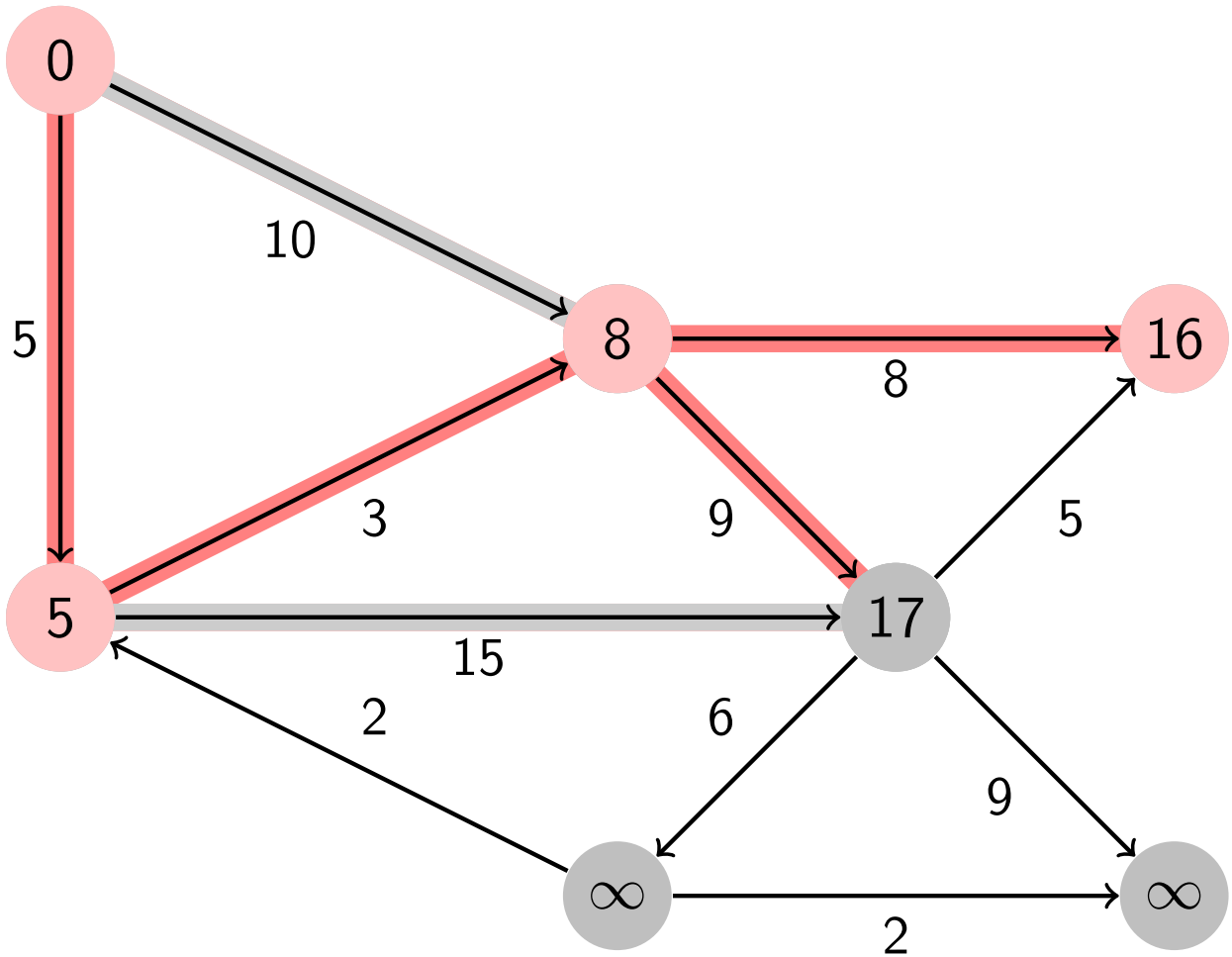
Dijkstra's algorithm



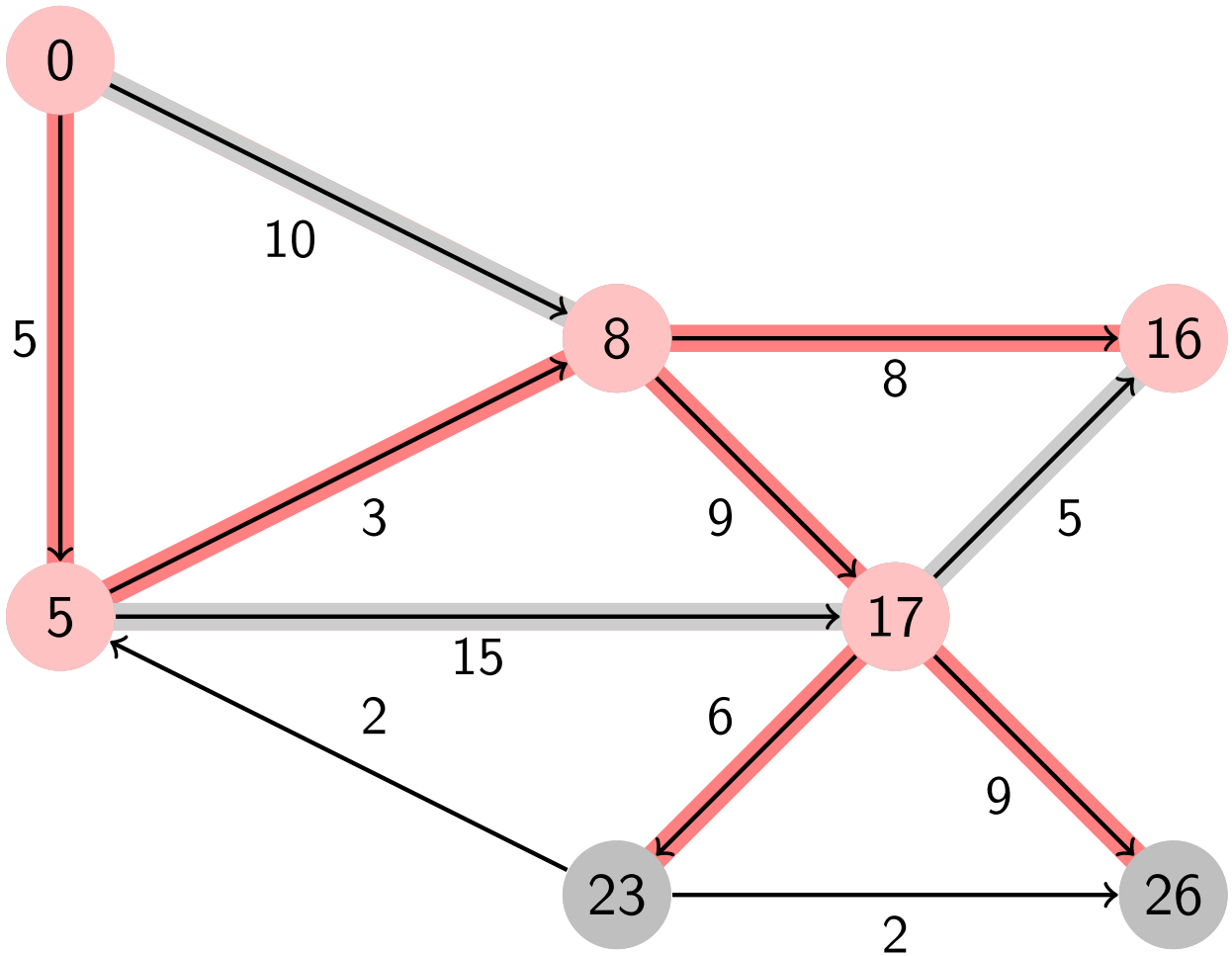
Dijkstra's algorithm



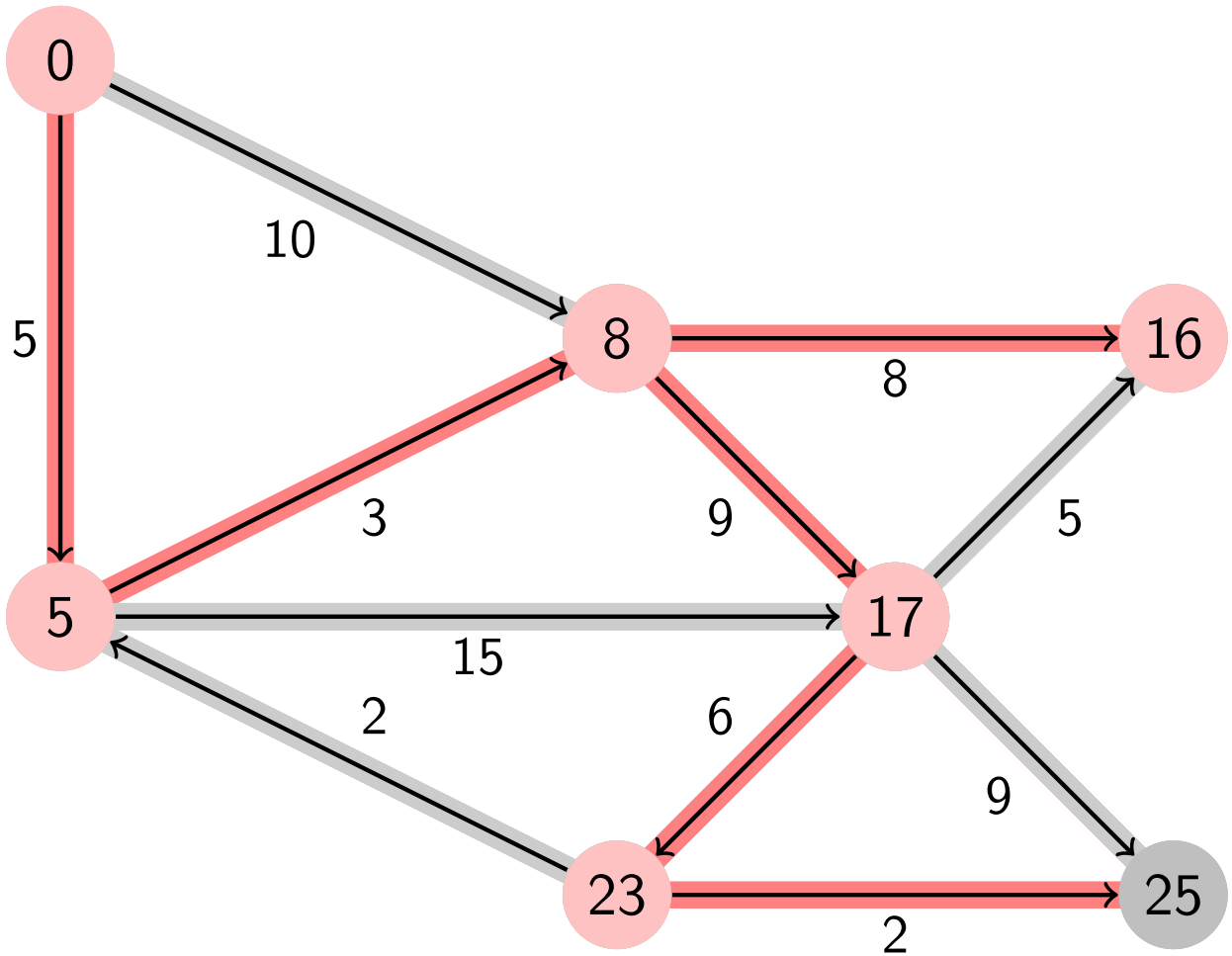
Dijkstra's algorithm



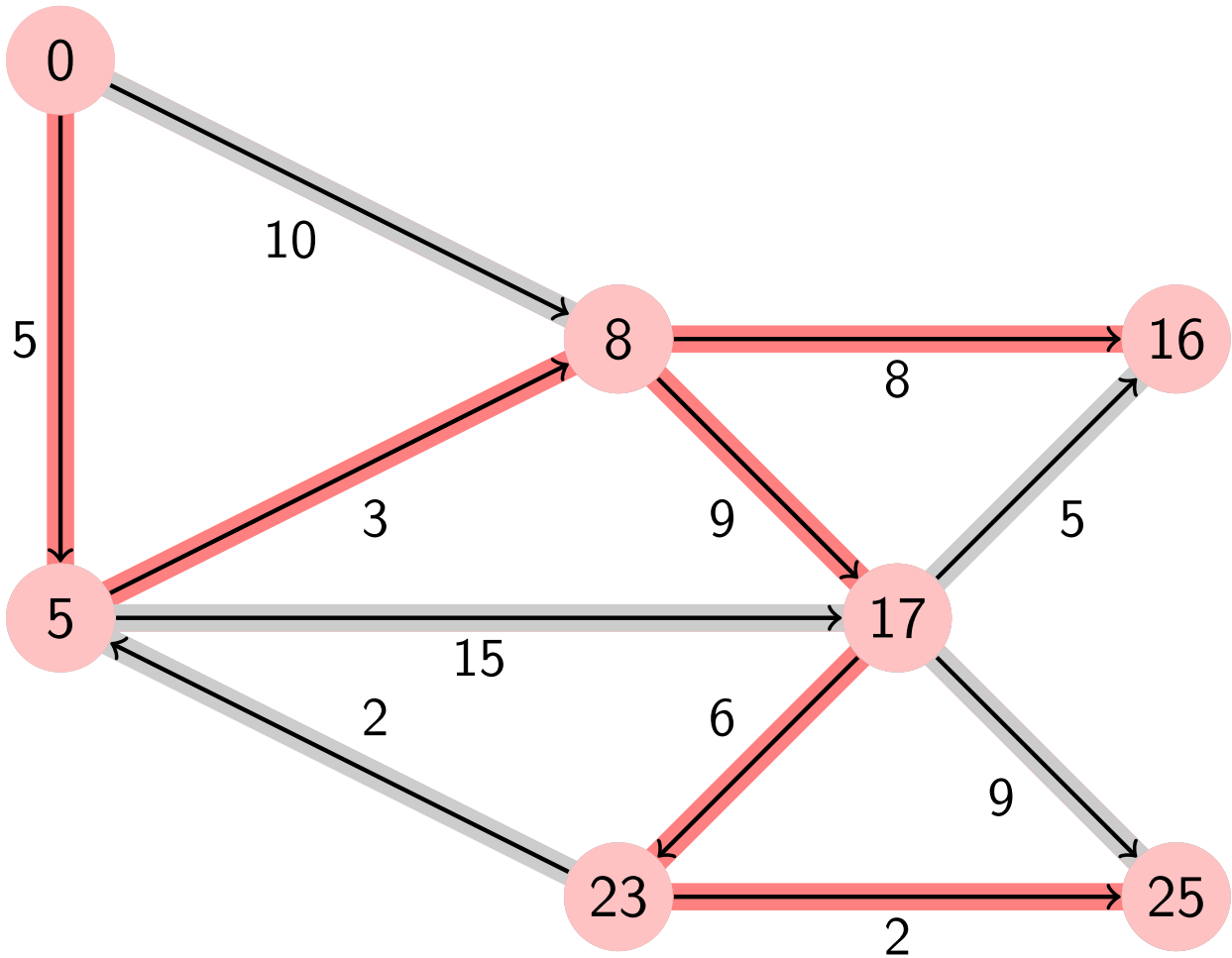
Dijkstra's algorithm



Dijkstra's algorithm



Dijkstra's algorithm



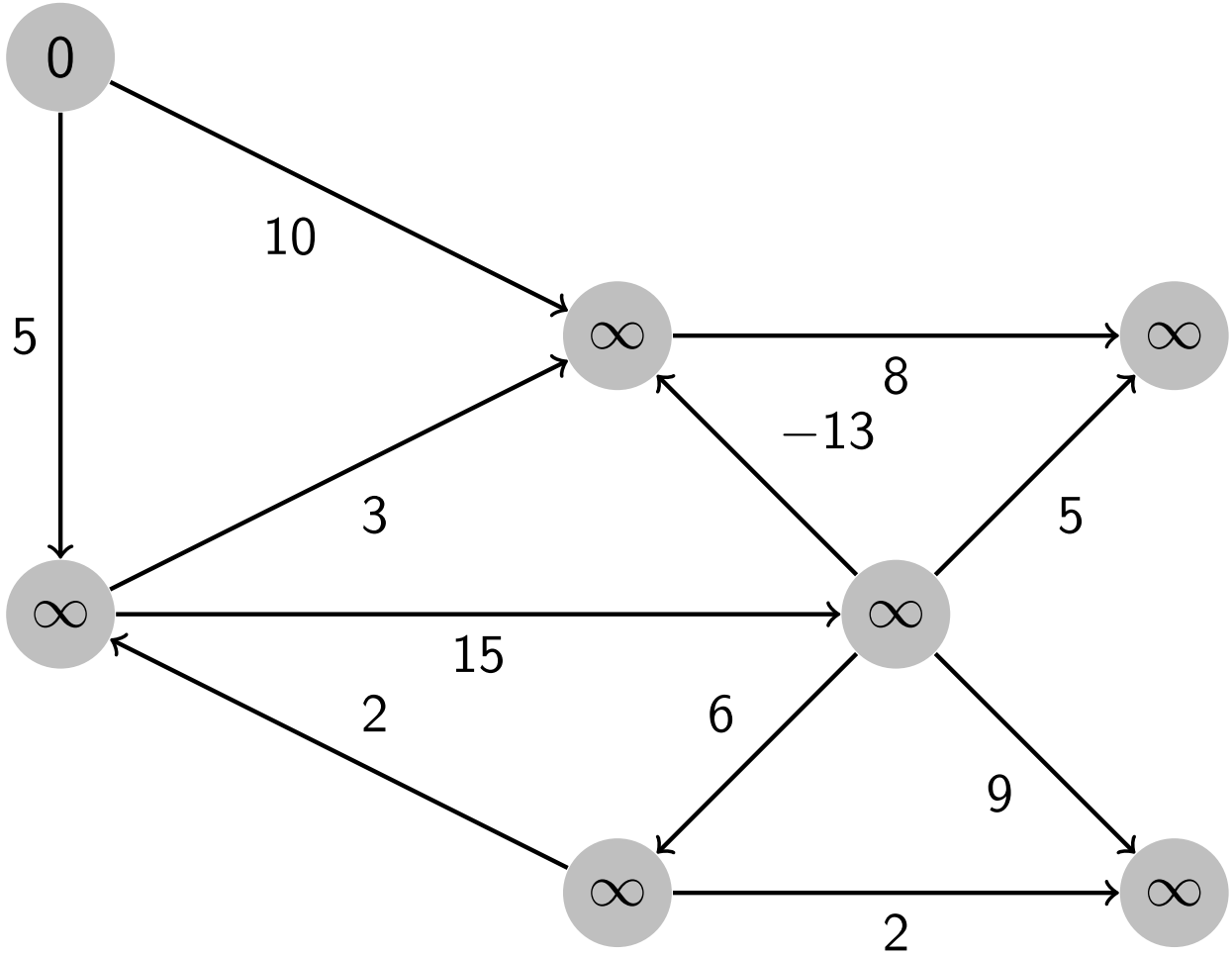
Moore-Bellman-Ford Algorithmus

Input: Ein Digraph D, eine konservative Funktion $c : A(D) \to \mathbb{R}$ und $s \in V(D)$.

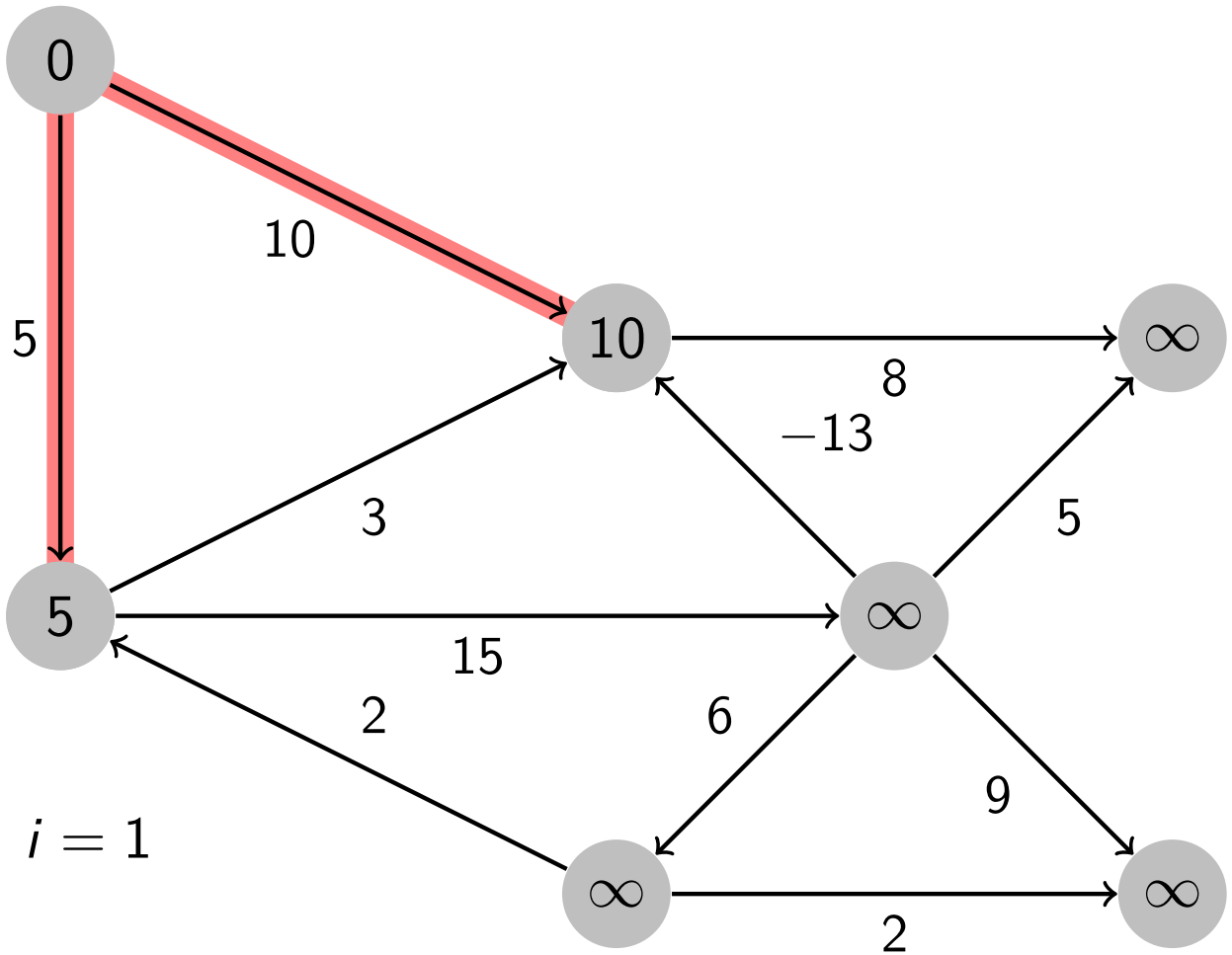
Output: Wie beim Dijkstra's Algorithm: $(l(v), p(v))_{v \in V(D) \setminus \{r\}}$ wobei $l(v) = dist_{(D,c)}(s, v)$ gilt und $p(v)$ ein Vorgänger von v auf einem kürzesten Weg von s nach v ist. (Ist v nicht erreichbar, so ist $l(v) = \infty$ und $p(v)$ nicht definiert.)

begin
  $l(s) \leftarrow 0$;
  for $v \in V(D) \setminus \{s\}$ **do**
    $l(v) \leftarrow \infty$;
  end
  for $i = 1$ **to** $n(D) - 1$ **do**
    for $(v, w) \in A(D)$ **do**
      if $l(w) > l(v) + c((v, w))$ **then**
        $l(w) \leftarrow l(v) + c((v, w))$; $p(w) \leftarrow v$;
      end
    end
  end
  return $(l(v), p(v))_{v \in V(D) \setminus \{s\}}$;
end

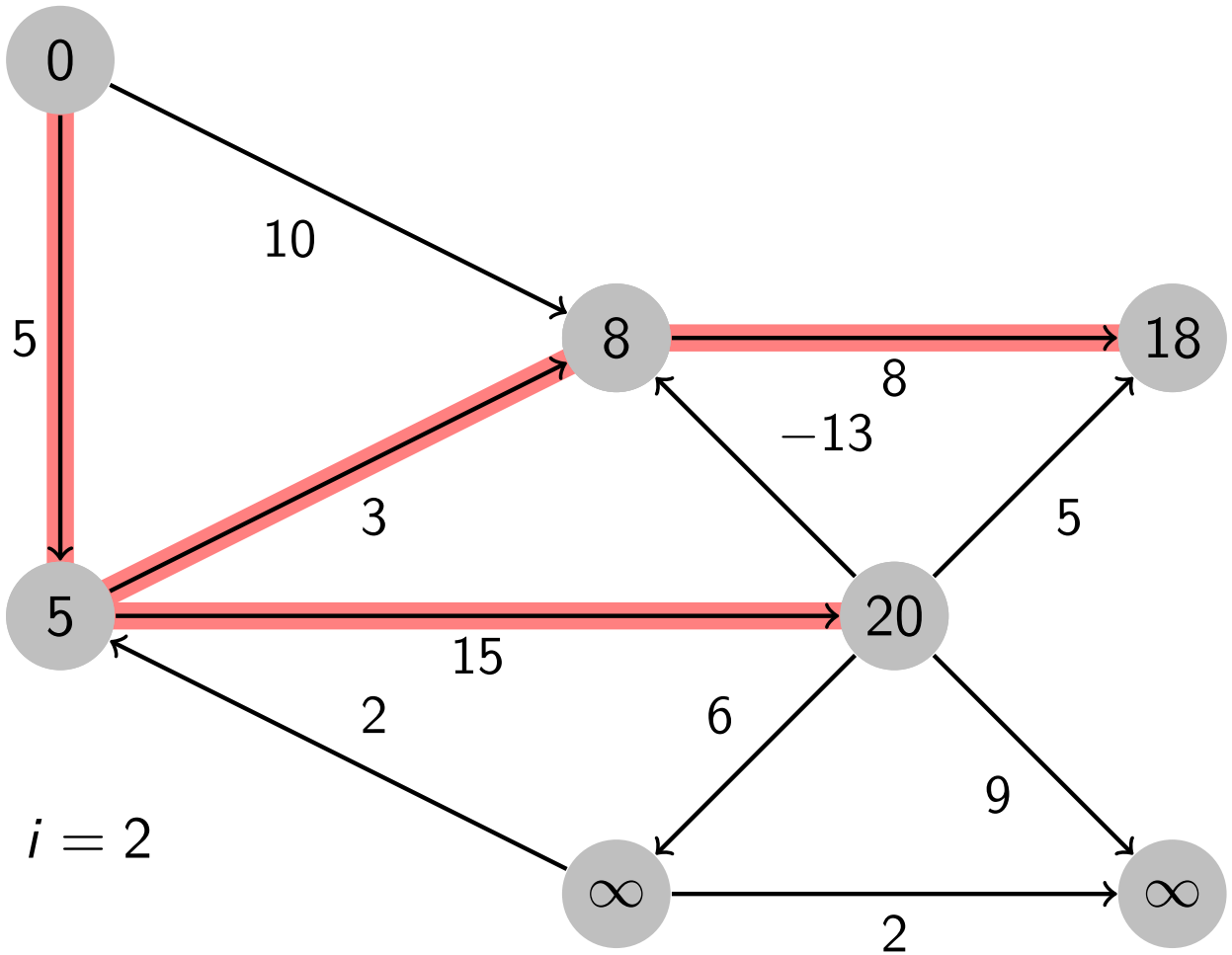
Moore-Bellman-Ford Algorithmus



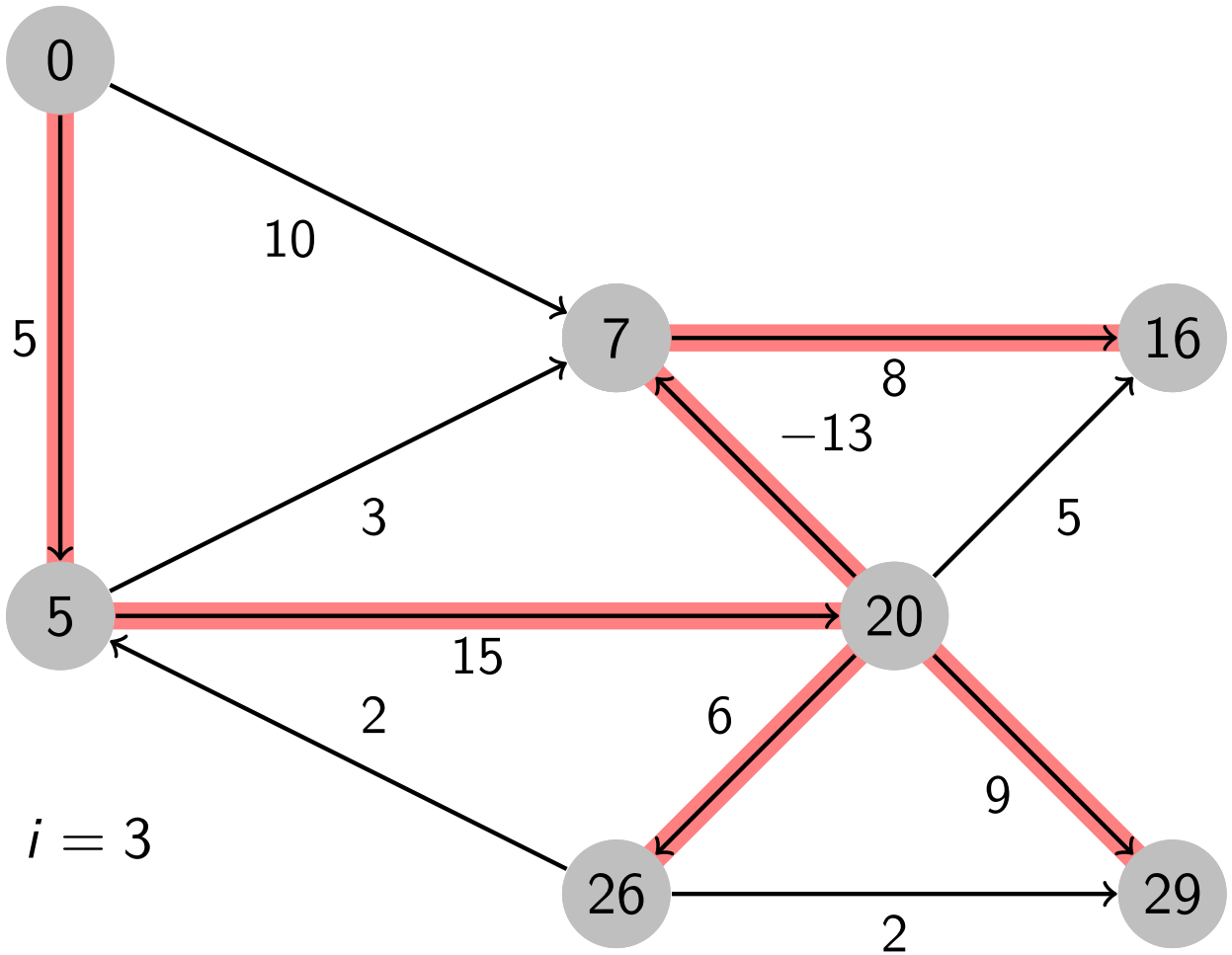
Moore-Bellman-Ford Algorithmus



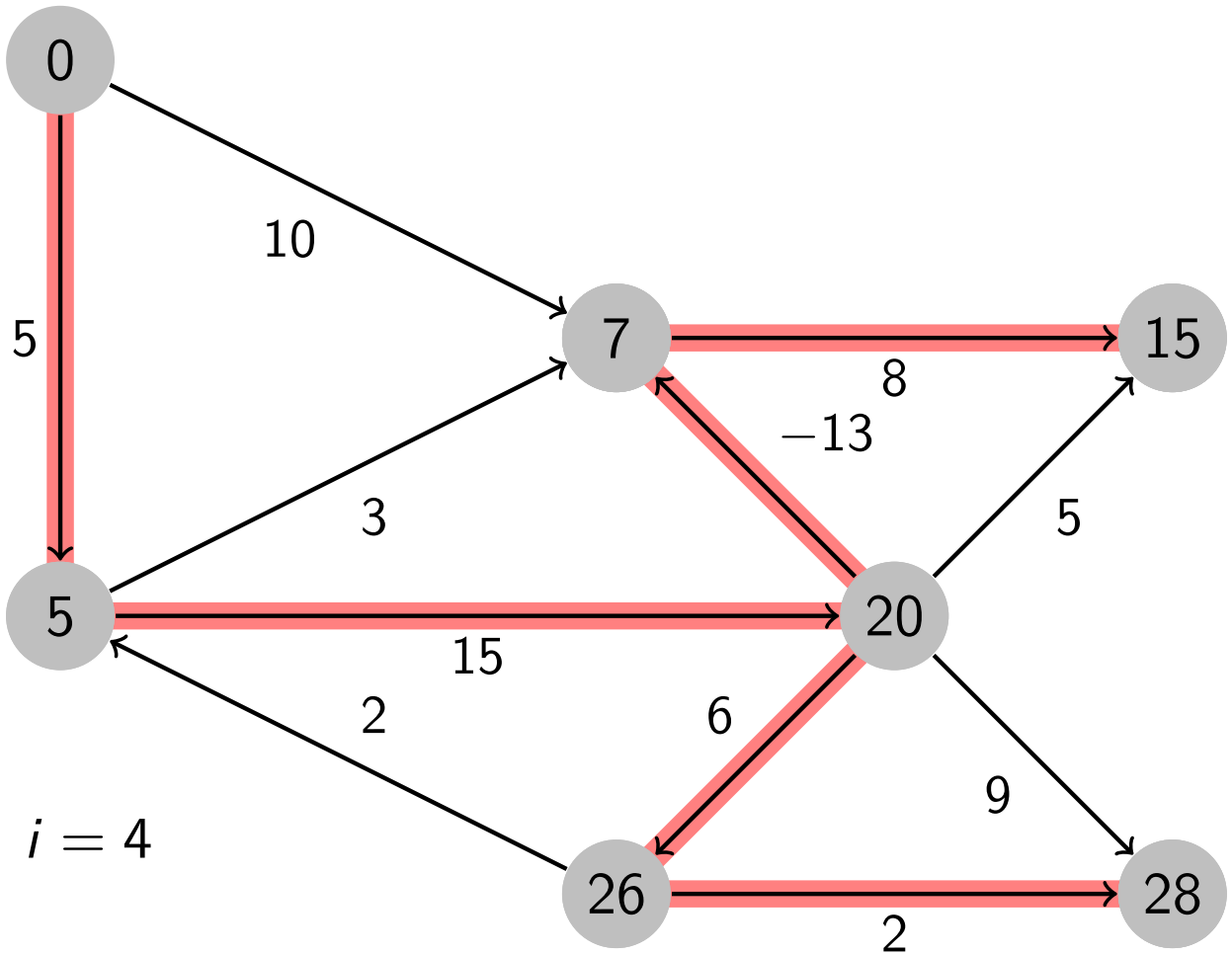
Moore-Bellman-Ford Algorithmus



Moore-Bellman-Ford Algorithmus



Moore-Bellman-Ford Algorithmus



Floyd-Warshall Algorithmus

Input: Ein Digraph D mit $V(D) = [n]$ und eine konservative Gewichtsfunktion $c : A(D) \to \mathbb{R}$.

Output: Matrizen $(l(i,j))_{(i,j)\in[n]^2}$ und $(p(i,j))_{(i,j)\in[n]^2}$. Hier bei gilt $l(i,j) = \mathrm{dist}_{(D,c)}(i,j)$ und für $(i,j) \in [n^2]$ mit $l(i,j) < \infty$ ist $(p(i,j),j)$ die letzte gerichte Kante eines bezüglich c kürzesten gerichteten Weges in D von i nach j.

Floyd-Warshall Algorithmus

begin
- **for** $(i,j) \in A(D)$ **do** $l(i,j) \leftarrow c((i,j))$; $p(i,j) \leftarrow i$;
- **for** $i,j \in V(D)$ *mit* $i \neq j$ *und* $(i,j) \notin A(D)$ **do** $l(i,j) \leftarrow \infty$;
- **for** $i \in V(D)$ **do** $l(i,i) \leftarrow 0$;
- **for** $j = 1$ **to** n **do**
  - **for** $i = 1$ **to** n **do**
    - **if** $i \neq j$ **then**
      - **for** $k = 1$ **to** n **do**
        - **if** $k \neq j$ **then**
          - **if** $l(i,k) > l(i,j) + l(j,k)$ **then**
            - $l(i,k) \leftarrow l(i,j) + l(j,k)$; $p(i,k) \leftarrow p(j,k)$;
          - **end**
        - **end**
      - **end**
    - **end**
  - **end**
- **end**

end