Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

# Übungen zu Graphen und Diskrete Optimierung

## Optimierung unter Ressourcenbeschränkung

Kendra Reiter
Basierend auf Folien von Alexander Wolff

Sommersemester 2023

# Organisatorisches

Zwei Übungsgruppen:

- Freitags, 8:00 - 10:00, Übungsraum 1
- Freitags, 14:00 - 16:00, Seminarraum 1

Nicht vergessen: Anmeldung über WueCampus um Zugriff auf Übungsblätter und Abgabe zu erhalten!

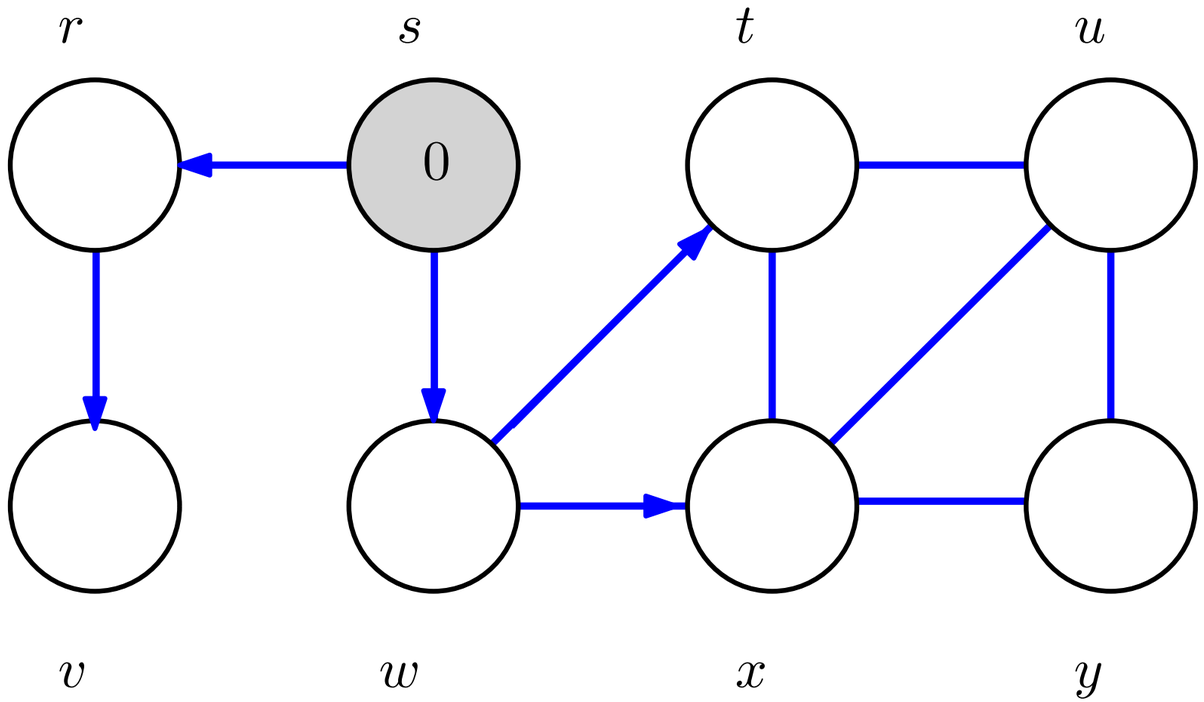
Neue Übungsblätter: jeden Mittwoch nach der Vorlesung

**Abgabe**: Mittwochs, 10:00 (**vor** der Vorlesung!)

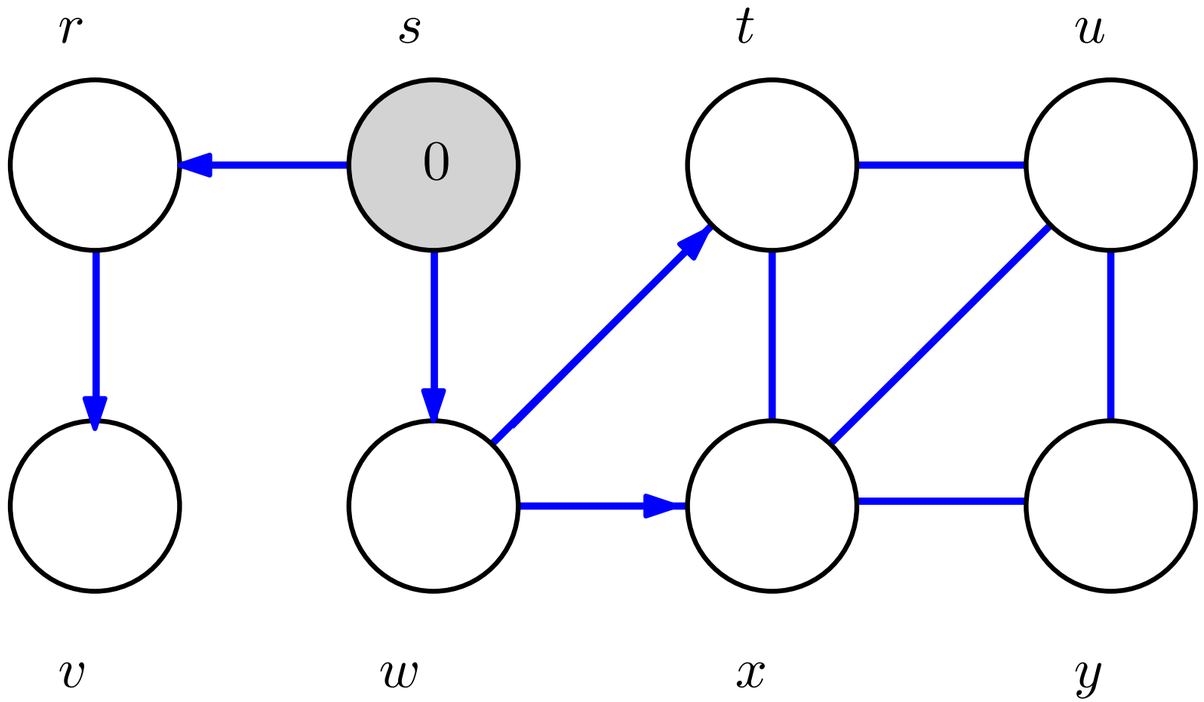
# Grundlegende Algorithmen aus der Vorlesung ADS

1. Graphendurchlaufstrategien
   - Breitensuche (BFS)
   - Tiefensuche (DFS)
2. Kürzeste Wege
   - Breitensuche (BFS)
   - Dijkstra
3. Minimale Spannbäume
   - Jarník-Prim
   - Krustal

# Breitensuche (BFS)

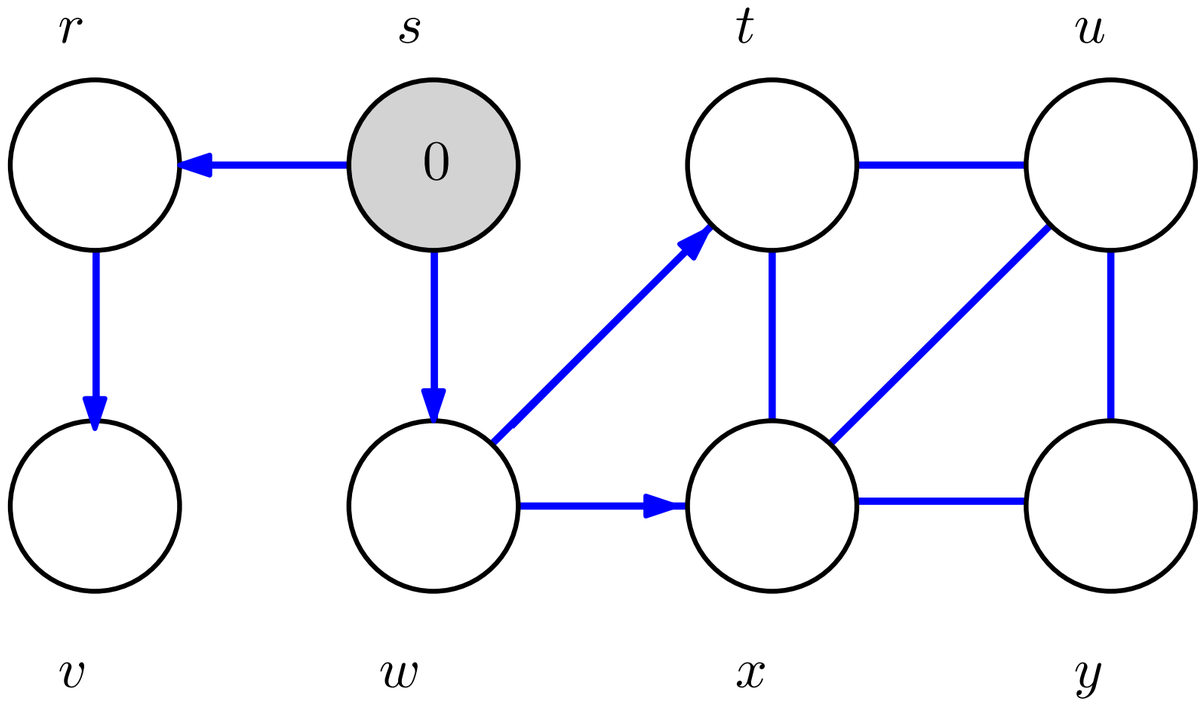


# Breitensuche (BFS)




Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

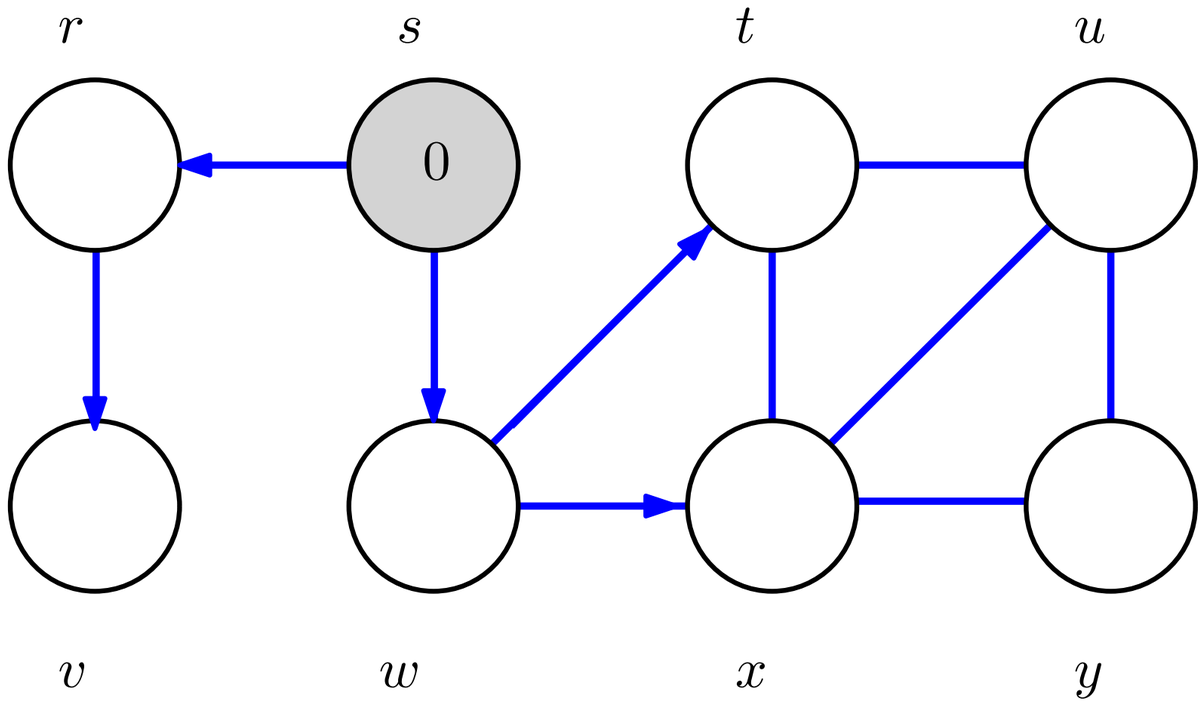
# Breitensuche (BFS)




Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**:

# Breitensuche (BFS)

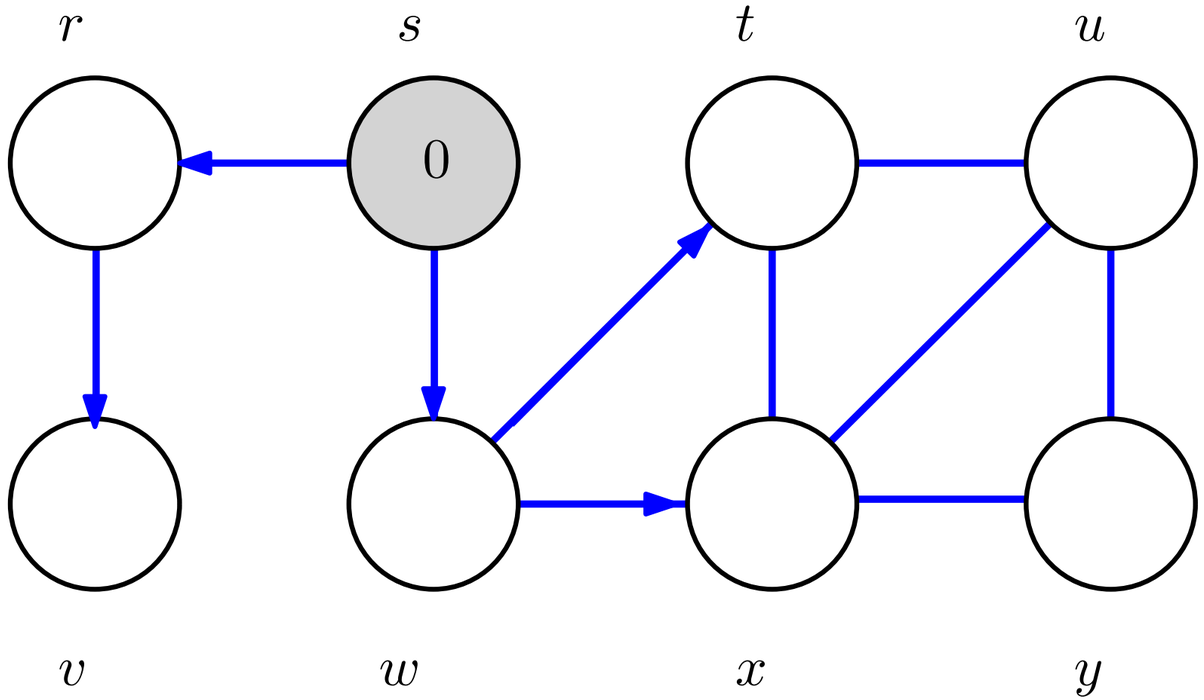



Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:**
**Output:**

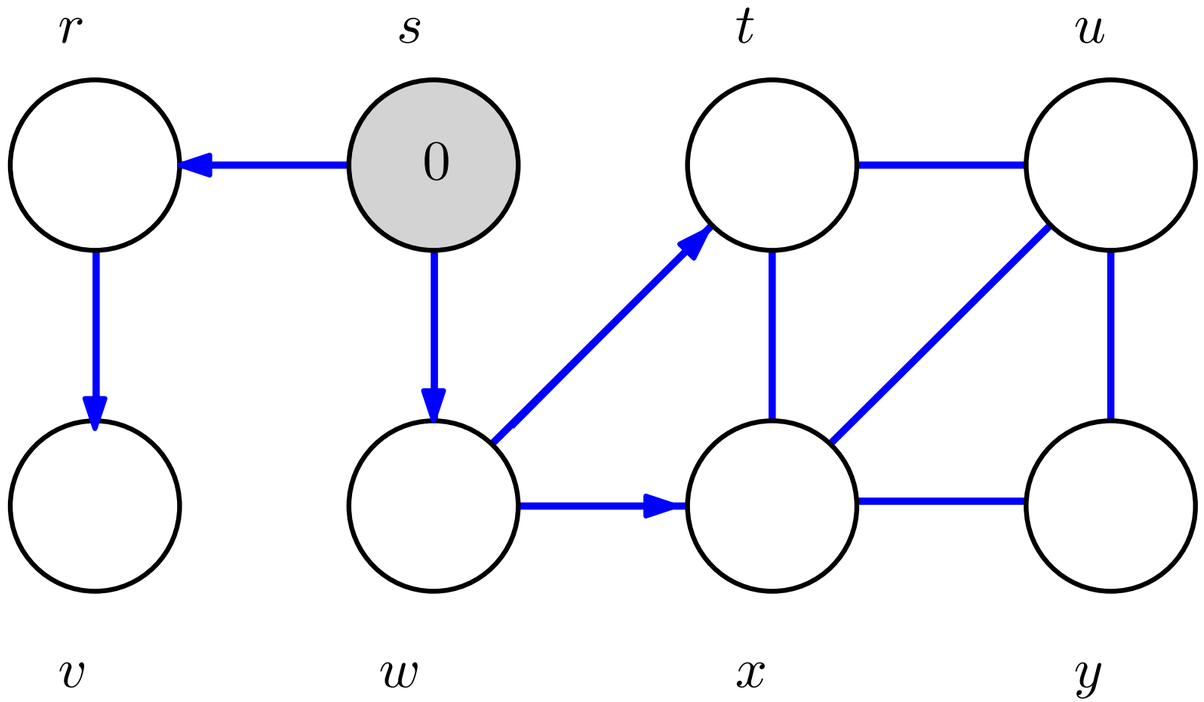



Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum



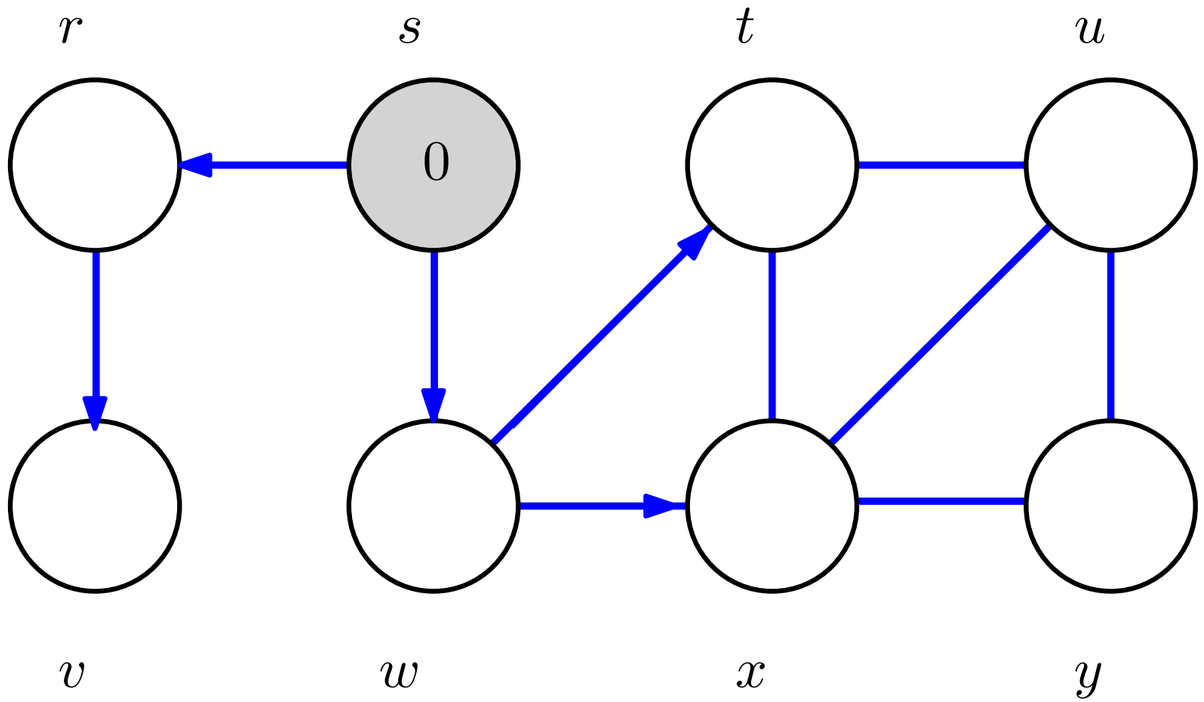

Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

$\textsc{Initialize}(G, s)$




**function** $\textsc{Initialize}(G, s)$
  **for** $u \in V$ **do**
    $u$.color $=$ white
    $u$.d $= \infty$
    $u.\pi =$ None
  **end for**
  $s$.color $=$ grey
  $s$.d $= 0$
**end function**

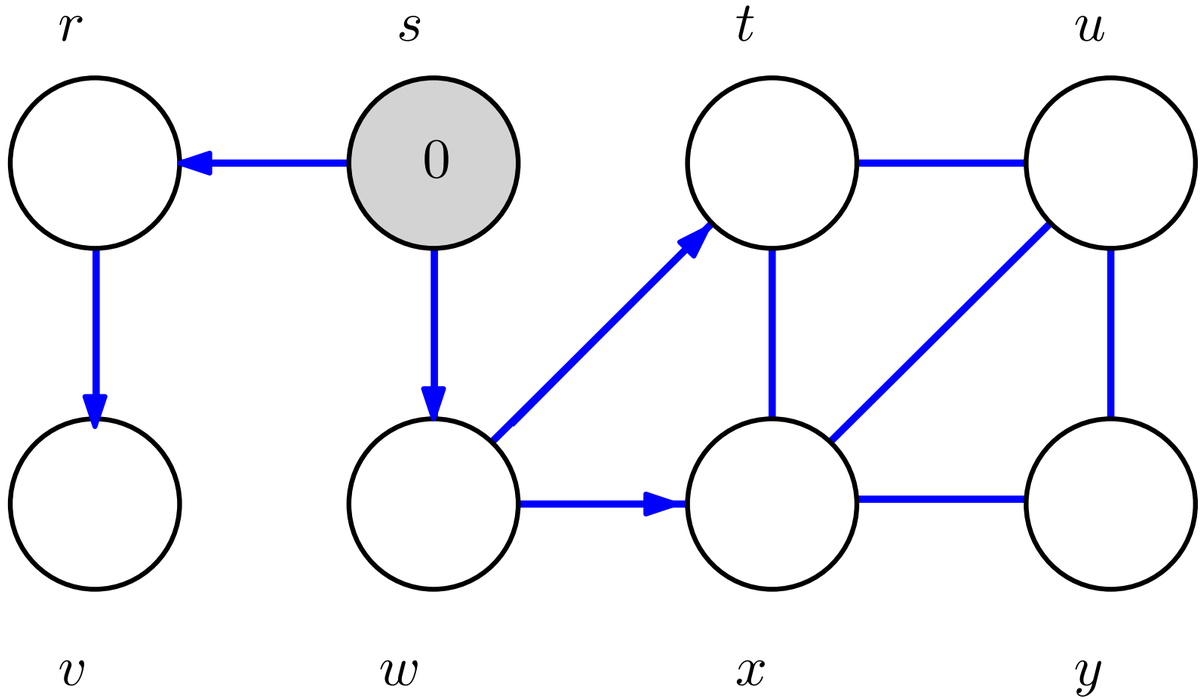
Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

INITIALIZE$(G, s)$
Q = new Queue()




**function** INITIALIZE$(G, s)$
  **for** $u \in V$ **do**
    $u$.color = white
    $u$.d = $\infty$
    $u.\pi$ = None
  **end for**
  $s$.color = grey
  $s$.d = 0
**end function**

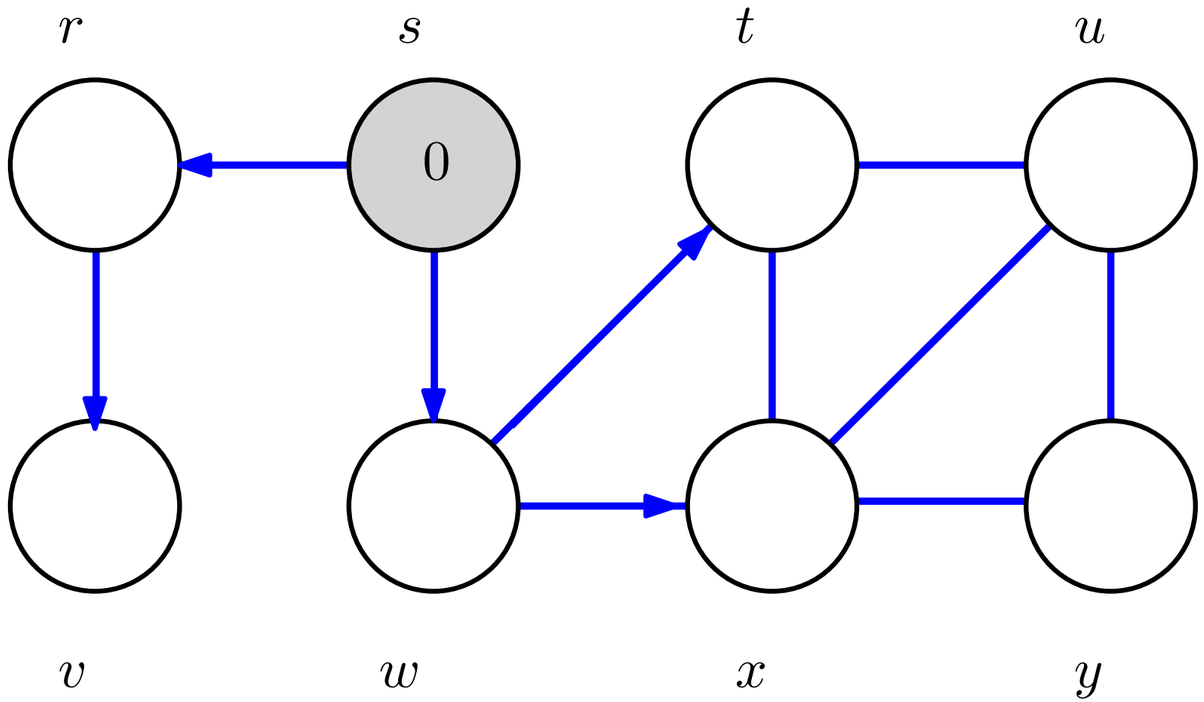
Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

INITIALIZE$(G, s)$
Q = new Queue()
Q.Enqueue($s$)




**function** INITIALIZE$(G, s)$
  **for** $u \in V$ **do**
    $u$.color = white
    $u$.d = $\infty$
    $u.\pi$ = None
  **end for**
  $s$.color = grey
  $s$.d = 0
**end function**

Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

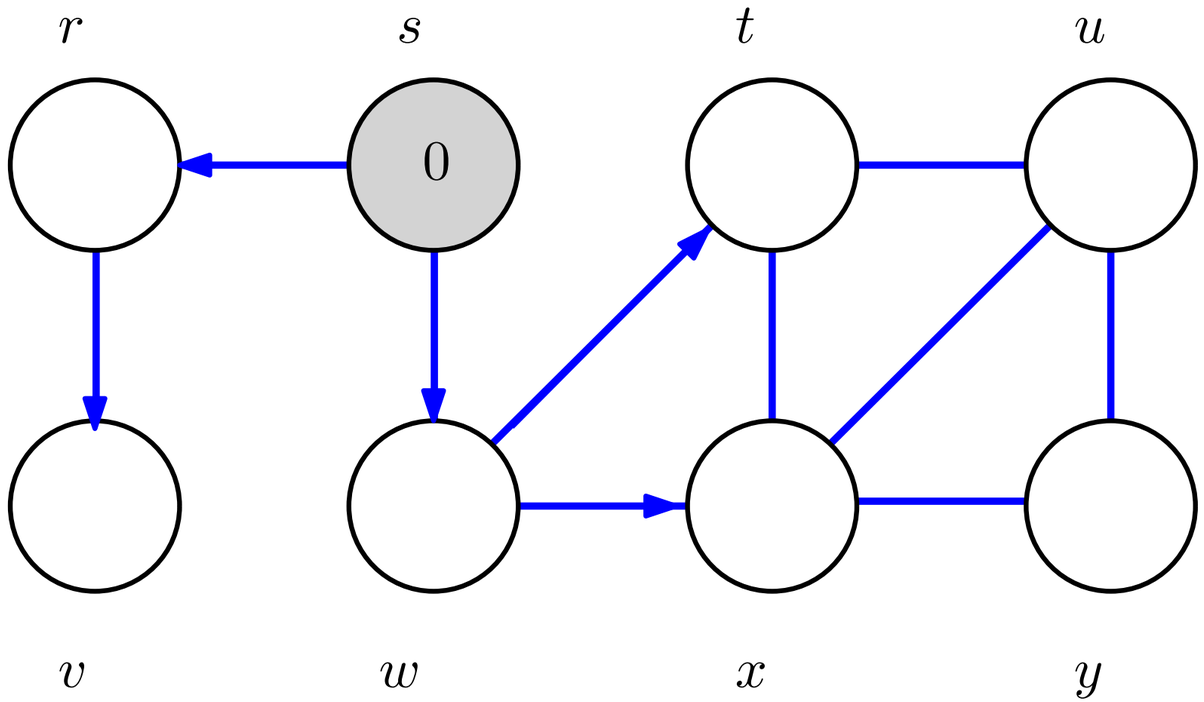
**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum
  INITIALIZE$(G, s)$
  Q = new Queue()
  Q.Enqueue($s$)
  **while** Q $\neq \emptyset$ **do**

  **end while**

**function** INITIALIZE$(G, s)$
  **for** $u \in V$ **do**
    $u$.color = white
    $u$.d $= \infty$
    $u.\pi$ = None
  **end for**
  $s$.color = grey
  $s$.d = 0
**end function**




Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

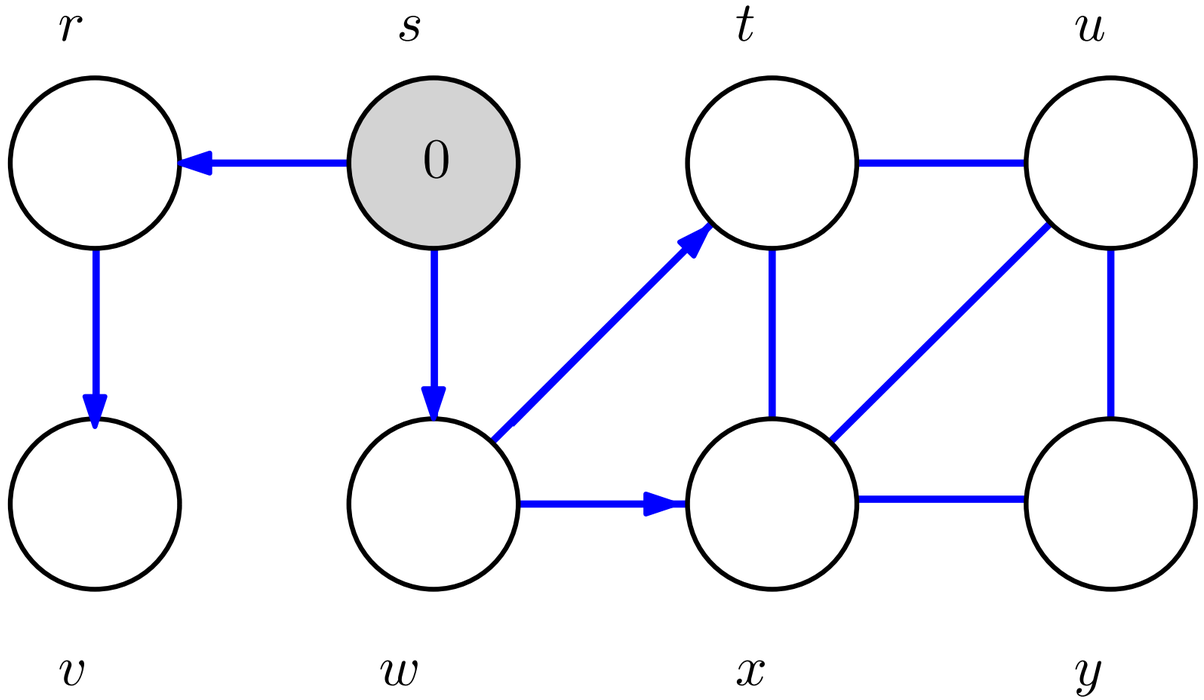
**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

```
Initialize(G, s)
Q = new Queue()
Q.Enqueue(s)
while Q ≠ ∅ do
    u = Q.Dequeue()

end while
```




```
function Initialize(G, s)
    for u ∈ V do
        u.color = white
        u.d = ∞
        u.π = None
    end for
    s.color = grey
    s.d = 0
end function
```

Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

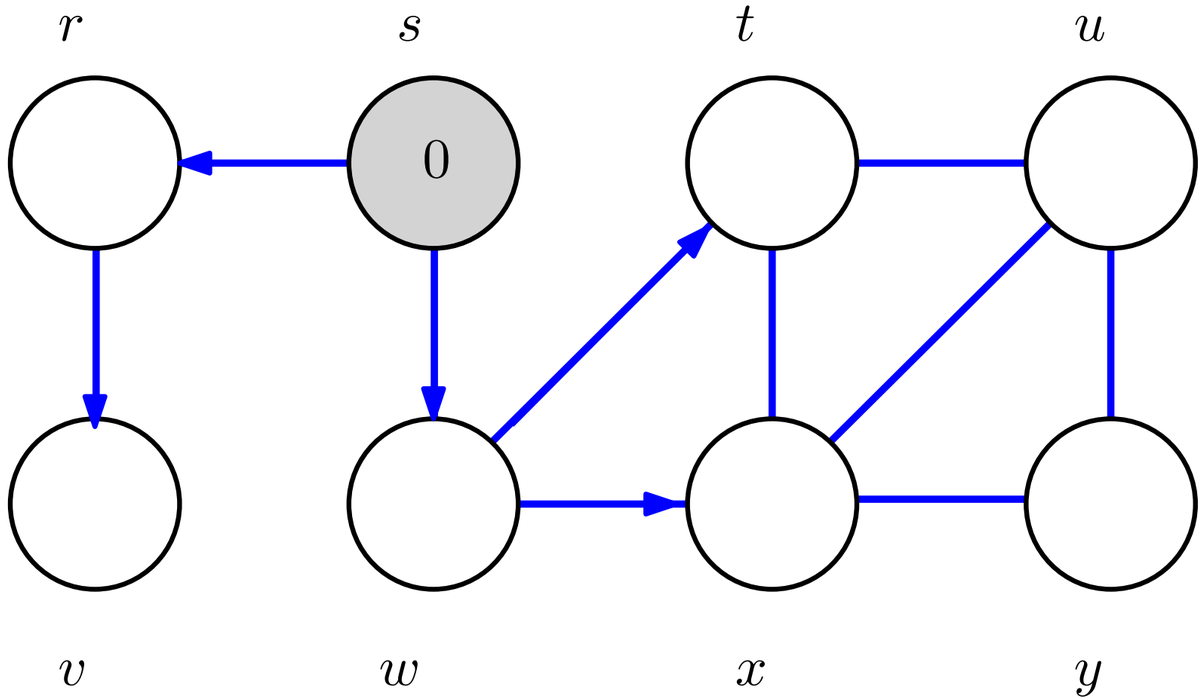
**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

```
Initialize(G, s)
Q = new Queue()
Q.Enqueue(s)
while Q ≠ ∅ do
    u = Q.Dequeue()
    for v ∈ Adj[u] do

    end for

end while
```

```
function Initialize(G, s)
    for u ∈ V do
        u.color = white
        u.d = ∞
        u.π = None
    end for
    s.color = grey
    s.d = 0
end function
```




Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

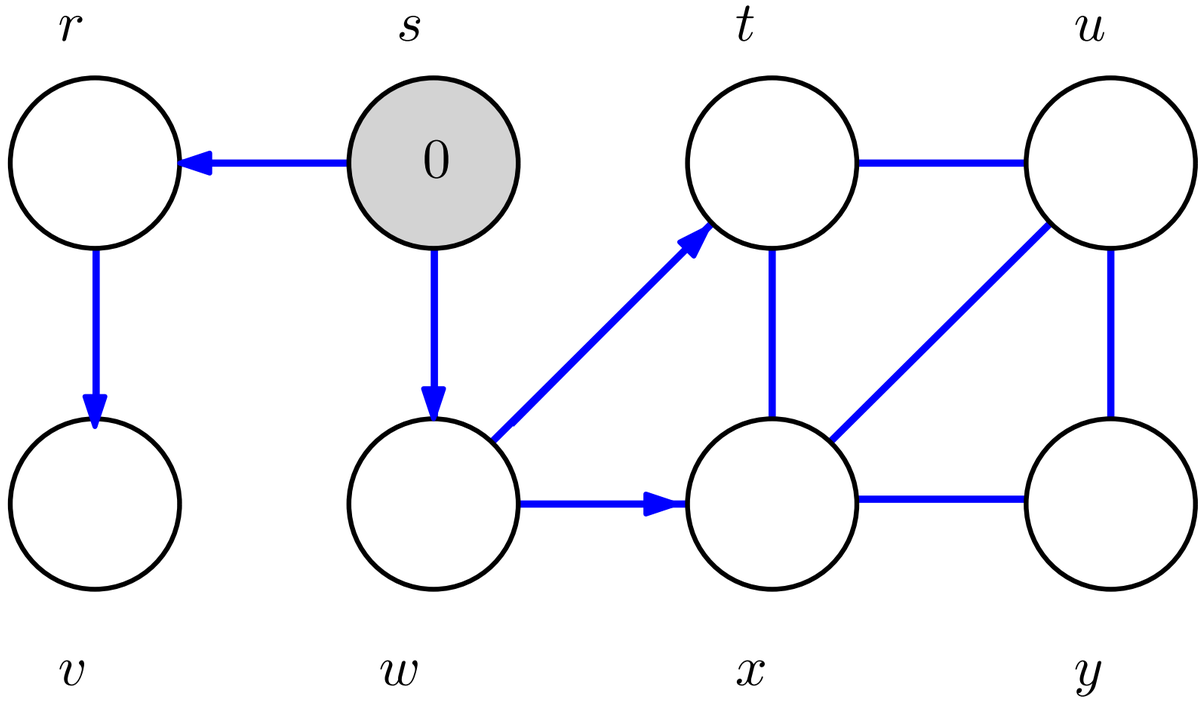
**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

```
Initialize(G, s)
Q = new Queue()
Q.Enqueue(s)
while Q ≠ ∅ do
    u = Q.Dequeue()
    for v ∈ Adj[u] do
        if v.color == white then

        end if
    end for

end while
```

```
function Initialize(G, s)
    for u ∈ V do
        u.color = white
        u.d = ∞
        u.π = None
    end for
    s.color = grey
    s.d = 0
end function
```




Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

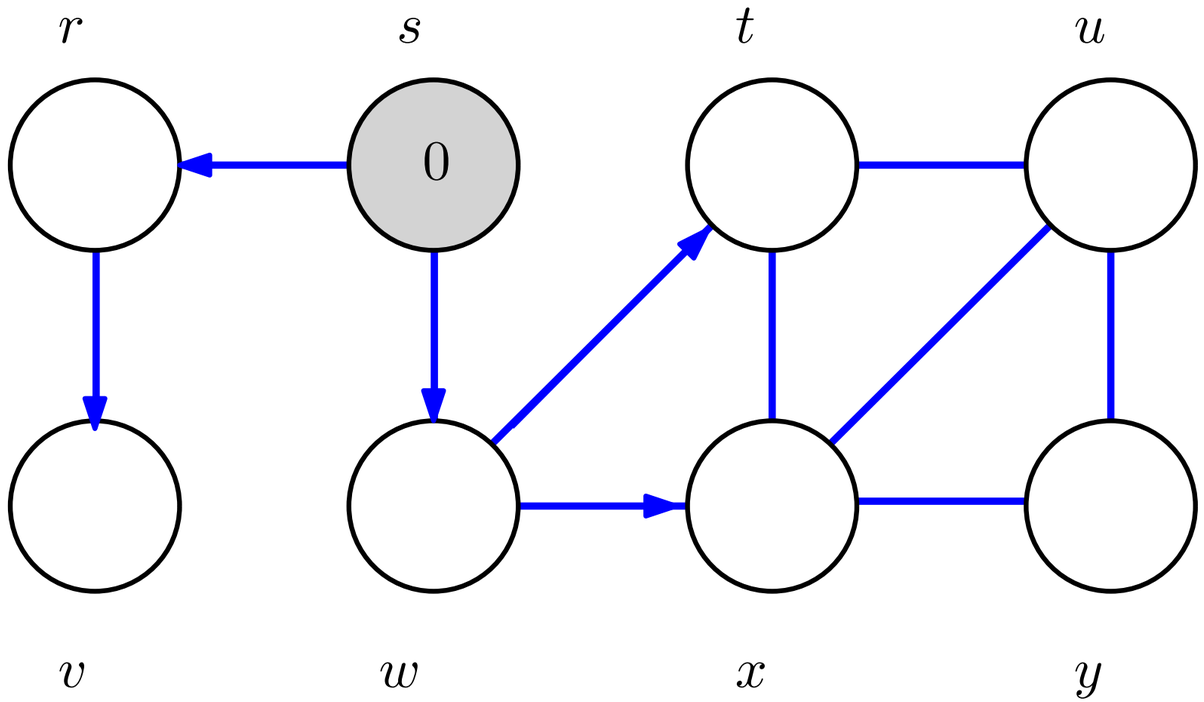
# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

```
INITIALIZE(G, s)
Q = new Queue()
Q.Enqueue(s)
while Q ≠ ∅ do
    u = Q.Dequeue()
    for v ∈ Adj[u] do
        if v.color == white then
            v.color = grey
            v.d = u.d + 1
            v.π = u
            Q.Enqueue(v)
        end if
    end for

end while
```

```
function INITIALIZE(G, s)
    for u ∈ V do
        u.color = white
        u.d = ∞
        u.π = None
    end for
    s.color = grey
    s.d = 0
end function
```




Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

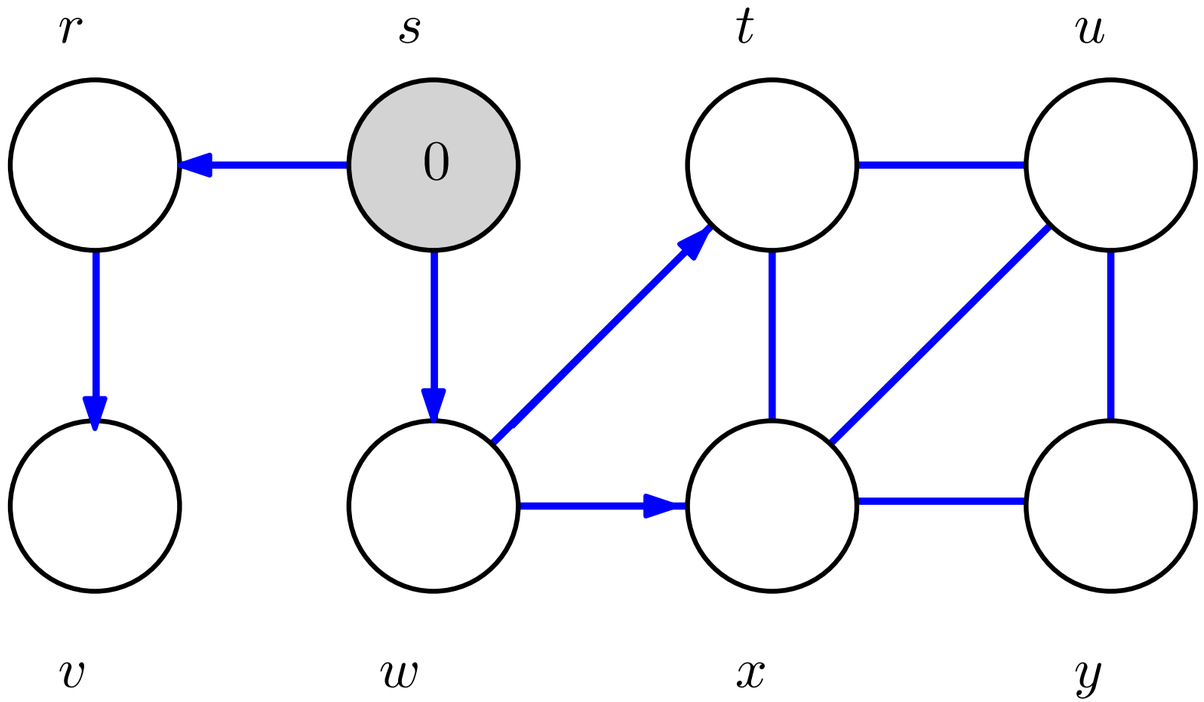
**Laufzeit**: $O(|V| + |E|)$

# Breitensuche (BFS)

**Input:** Graph $G$, Vertex $s$
**Output:** BFS-Baum

```
Initialize(G, s)
Q = new Queue()
Q.Enqueue(s)
while Q ≠ ∅ do
    u = Q.Dequeue()
    for v ∈ Adj[u] do
        if v.color == white then
            v.color = grey
            v.d = u.d + 1
            v.π = u
            Q.Enqueue(v)
        end if
    end for
    u.color = black
end while
```

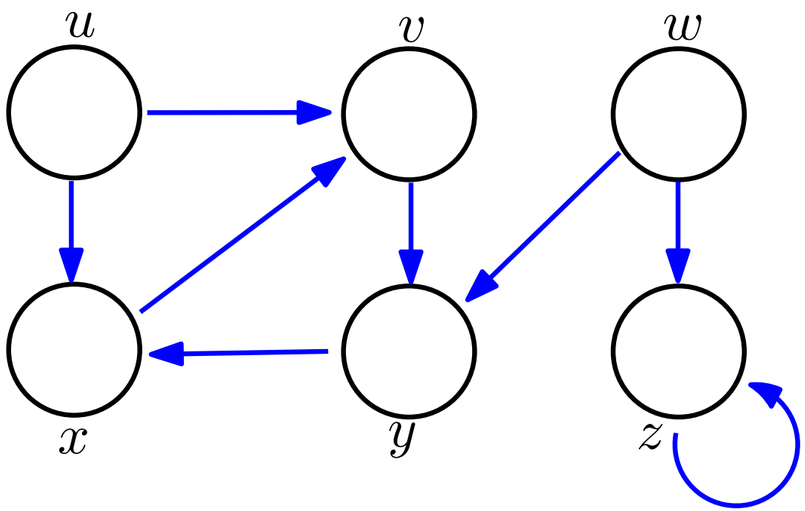
```
function Initialize(G, s)
    for u ∈ V do
        u.color = white
        u.d = ∞
        u.π = None
    end for
    s.color = grey
    s.d = 0
end function
```



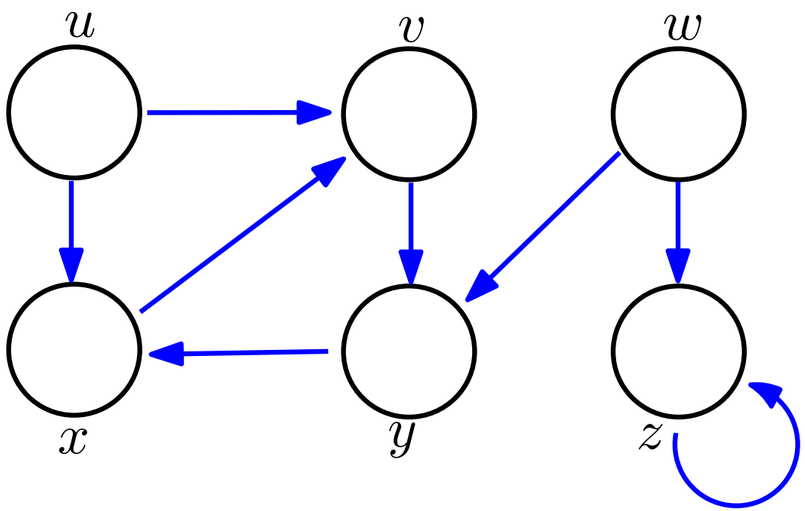

Suche durch Expansion in einzelnen Leveln des Graphen, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Tiefensuche (DFS)

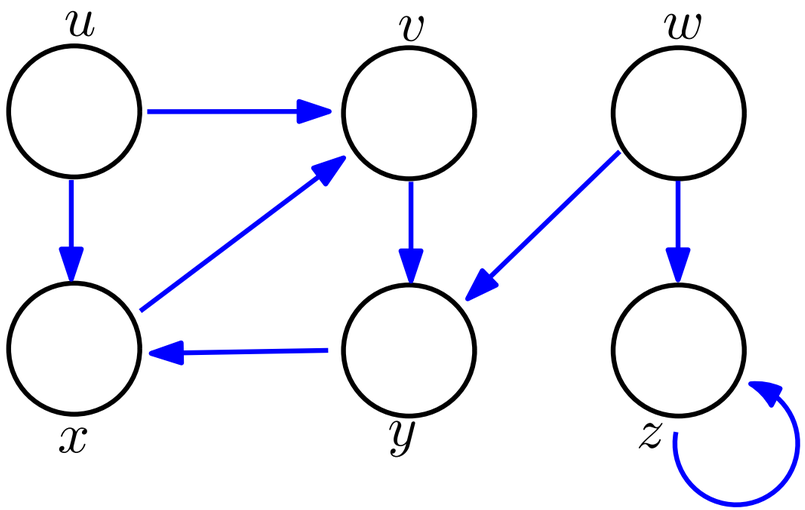


# Tiefensuche (DFS)




Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

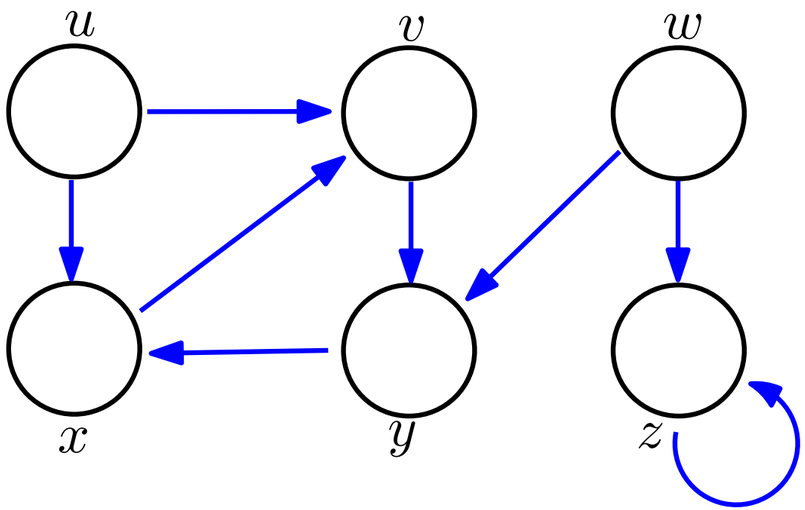
# Tiefensuche (DFS)




Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

**Laufzeit**:

# Tiefensuche (DFS)

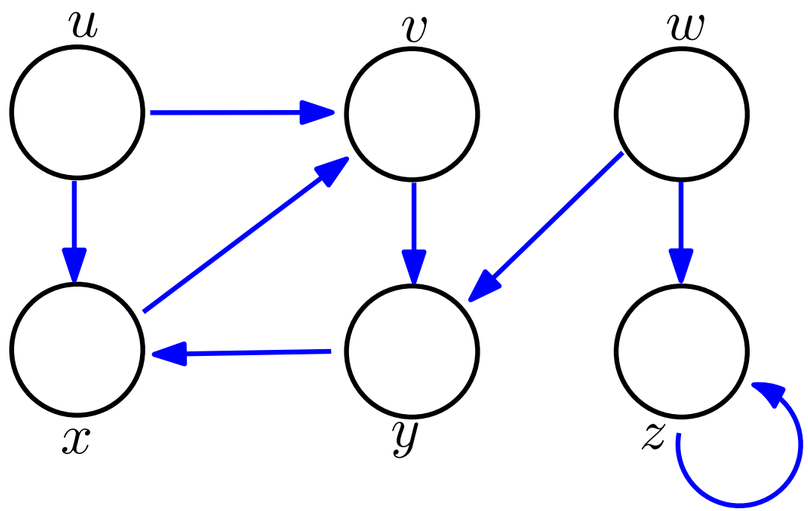



Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Tiefensuche (DFS)

**Input:**
**Output:**

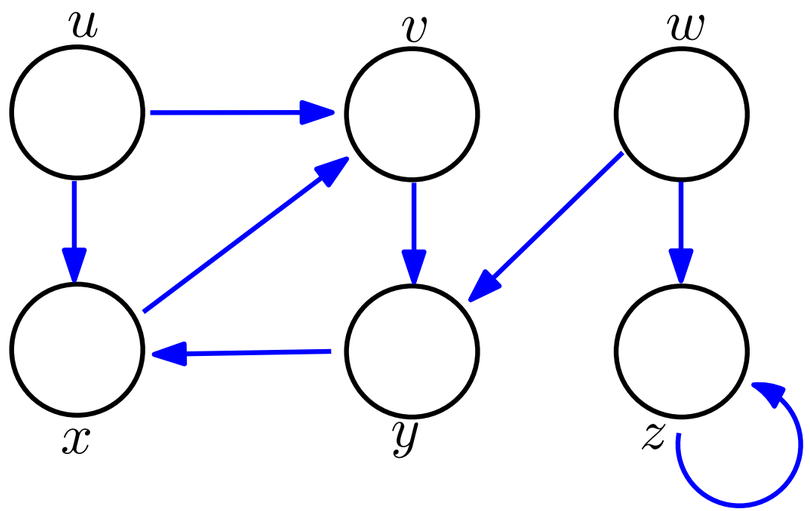



Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Tiefensuche (DFS)

**Input:** Graph $G = (V, E)$
**Output:** Besuchsintervalle, DFS-Wald



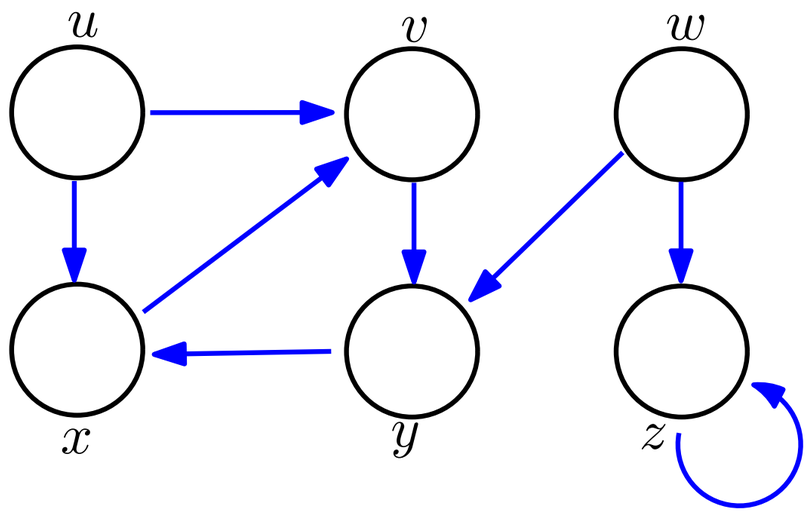

Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Tiefensuche (DFS)

**Input:** Graph $G = (V, E)$
**Output:** Besuchsintervalle, DFS-Wald

```
for u ∈ V do
    u.color = white
    u.π = None
end for
time = 0 //globale Variable
for u ∈ V do
    if u.color == white then
        DFSVisit(G, u)
    end if
end for
```




Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

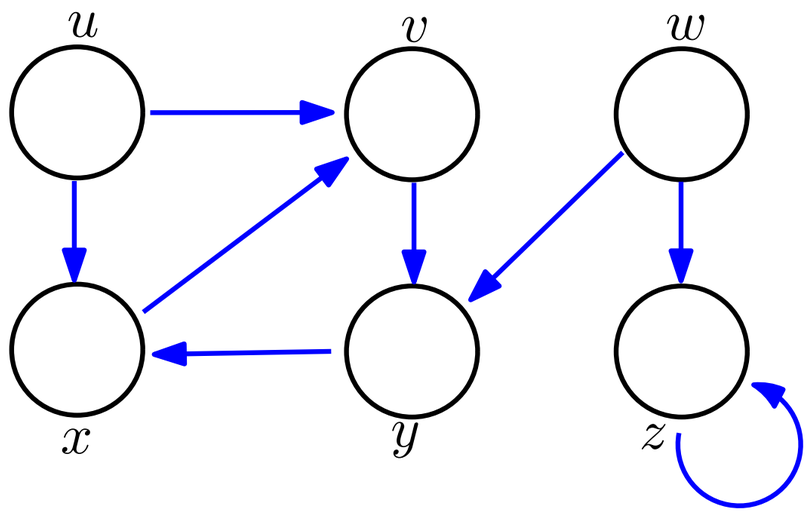
**Laufzeit**: $O(|V| + |E|)$

# Tiefensuche (DFS)

**Input:** Graph $G = (V, E)$
**Output:** Besuchsintervalle, DFS-Wald
  **for** $u \in V$ **do**
    $u$.color = white
    $u.\pi$ = None
  **end for**
  time = 0 //globale Variable
  **for** $u \in V$ **do**
    **if** $u$.color == white **then**
      DFSVisit($G$, $u$)
    **end if**
  **end for**

**function** DFSVisit(Graph $G$, Vertex $u$)
  time = time + 1
  $u$.d = time
  $u$.color = grey

**end function**




Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Tiefensuche (DFS)

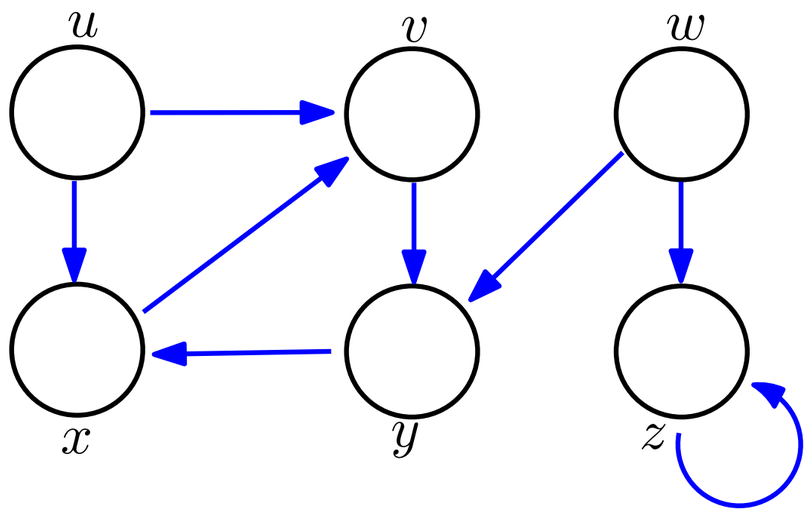
**Input:** Graph $G = (V, E)$
**Output:** Besuchsintervalle, DFS-Wald

```
for u ∈ V do
    u.color = white
    u.π = None
end for
time = 0 //globale Variable
for u ∈ V do
    if u.color == white then
        DFSVisit(G, u)
    end if
end for
```

```
function DFSVisit(Graph G, Vertex u)
    time = time + 1
    u.d = time
    u.color = grey
    for v ∈ Adj[u] do

    end for

end function
```




Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

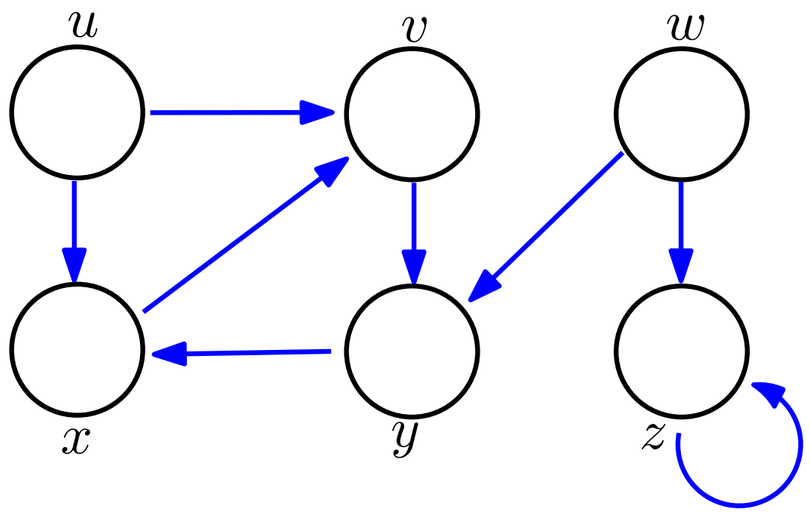
# Tiefensuche (DFS)

**Input:** Graph $G = (V, E)$
**Output:** Besuchsintervalle, DFS-Wald

```
for u ∈ V do
    u.color = white
    u.π = None
end for
time = 0 //globale Variable
for u ∈ V do
    if u.color == white then
        DFSVisit(G, u)
    end if
end for
```

```
function DFSVisit(Graph G, Vertex u)
    time = time + 1
    u.d = time
    u.color = grey
    for v ∈ Adj[u] do
        if v.color == white then
            v.π = u
            DFSVisit(G, v)
        end if
    end for

end function
```




Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

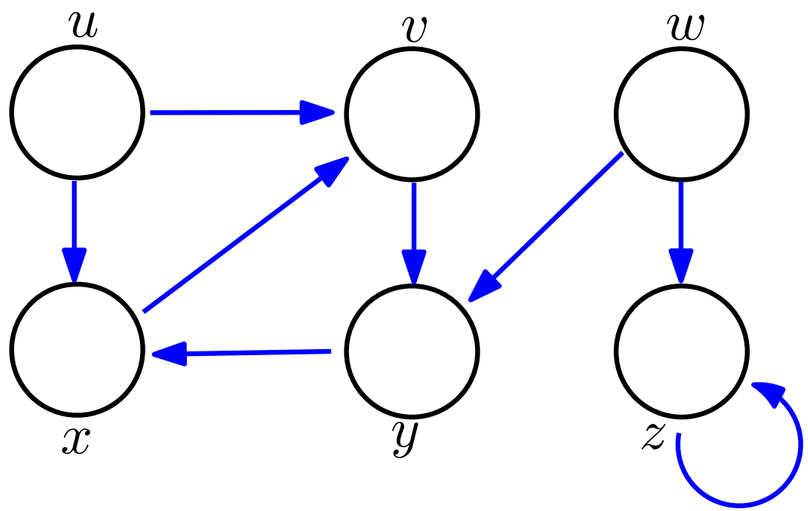
**Laufzeit**: $O(|V| + |E|)$

# Tiefensuche (DFS)

**Input:** Graph $G = (V, E)$
**Output:** Besuchsintervalle, DFS-Wald

```
for u ∈ V do
    u.color = white
    u.π = None
end for
time = 0 //globale Variable
for u ∈ V do
    if u.color == white then
        DFSVisit(G, u)
    end if
end for
```

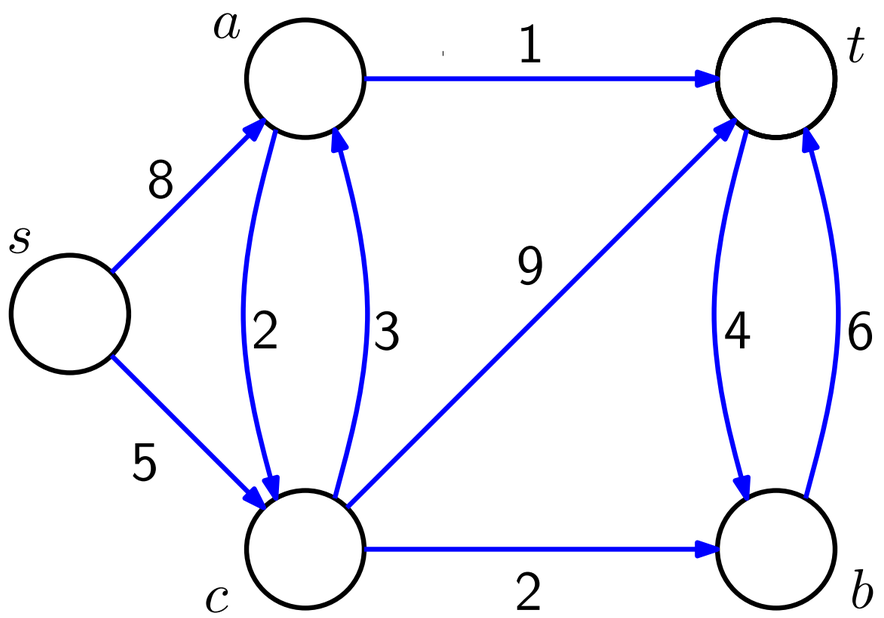
```
function DFSVisit(Graph G, Vertex u)
    time = time + 1
    u.d = time
    u.color = grey
    for v ∈ Adj[u] do
        if v.color == white then
            v.π = u
            DFSVisit(G, v)
        end if
    end for
    time = time + 1
    u.f = time
    u.color = black
end function
```



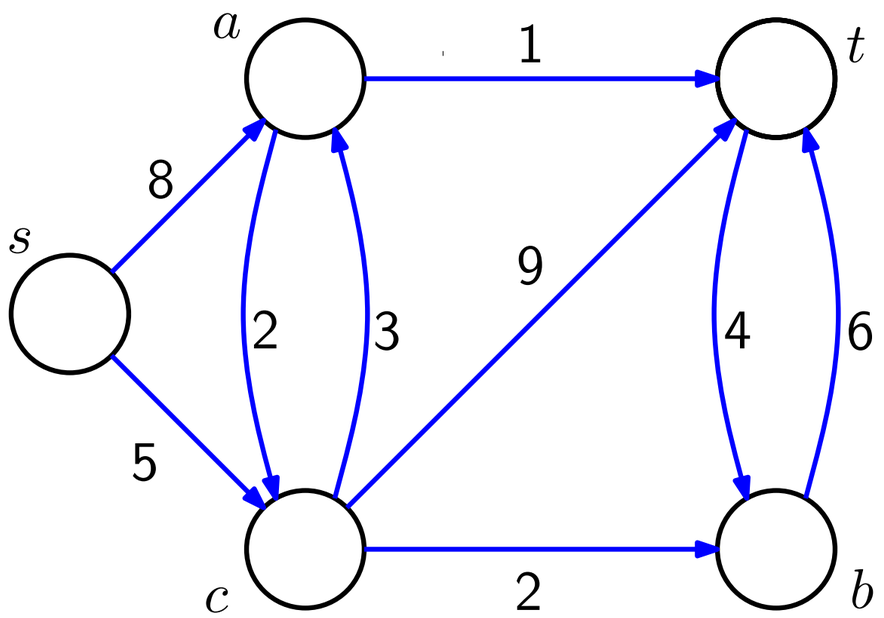

Suche durch Expansion nach und nach in allen Nachfolgeknoten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| + |E|)$

# Dijkstra

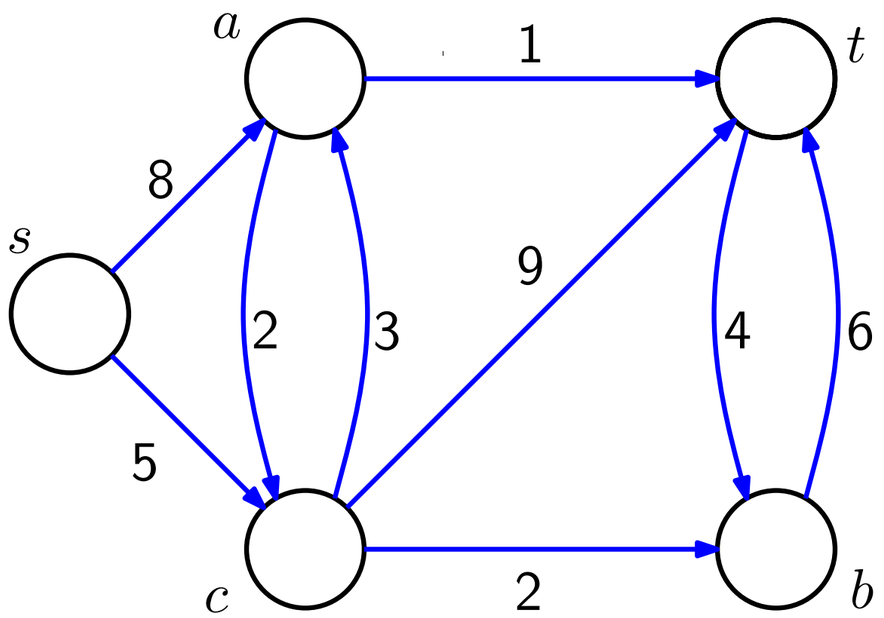


# Dijkstra




Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

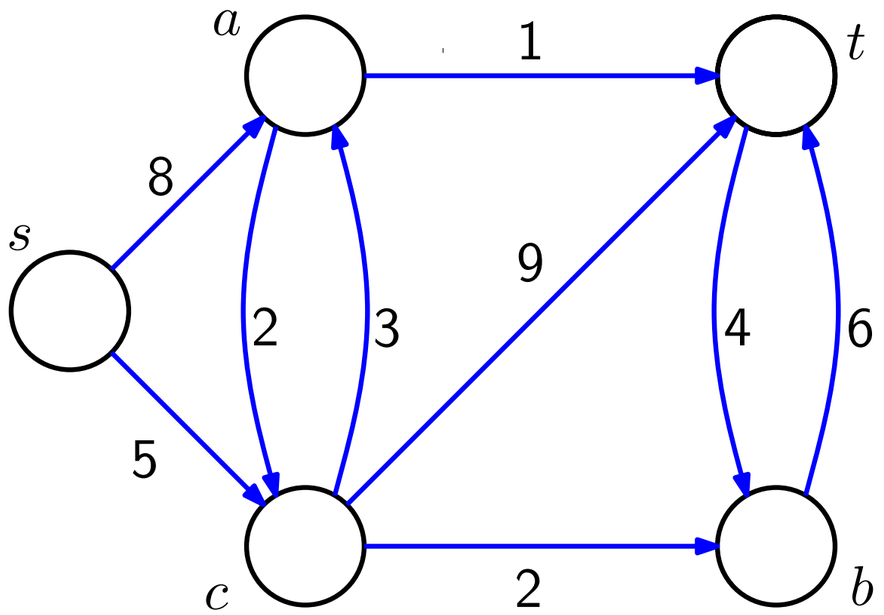
# Dijkstra




Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**:

# Dijkstra

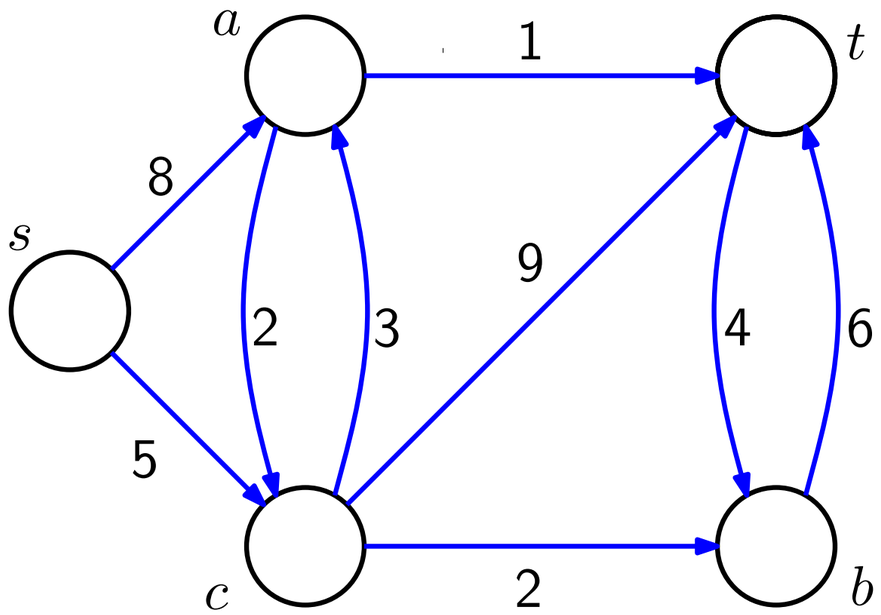



Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:**
**Output:**

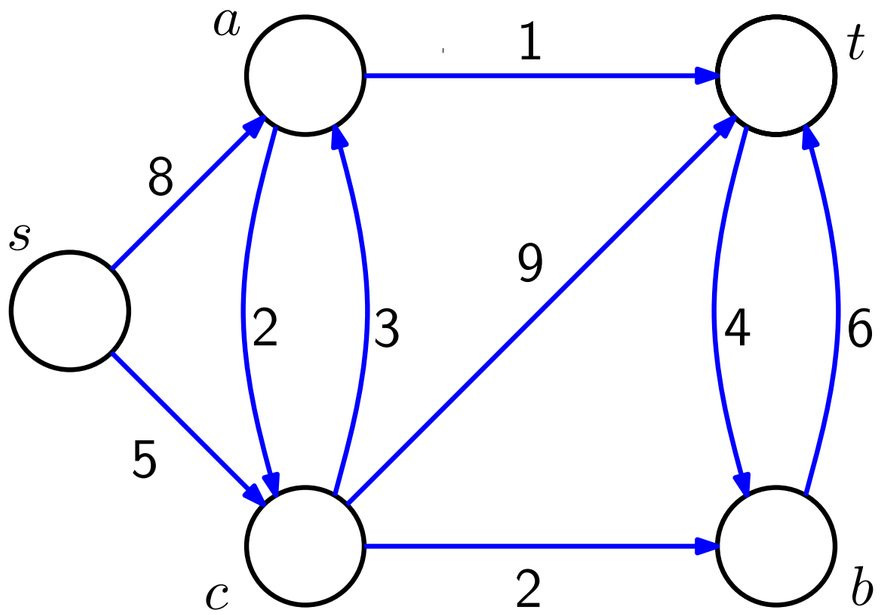



Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$

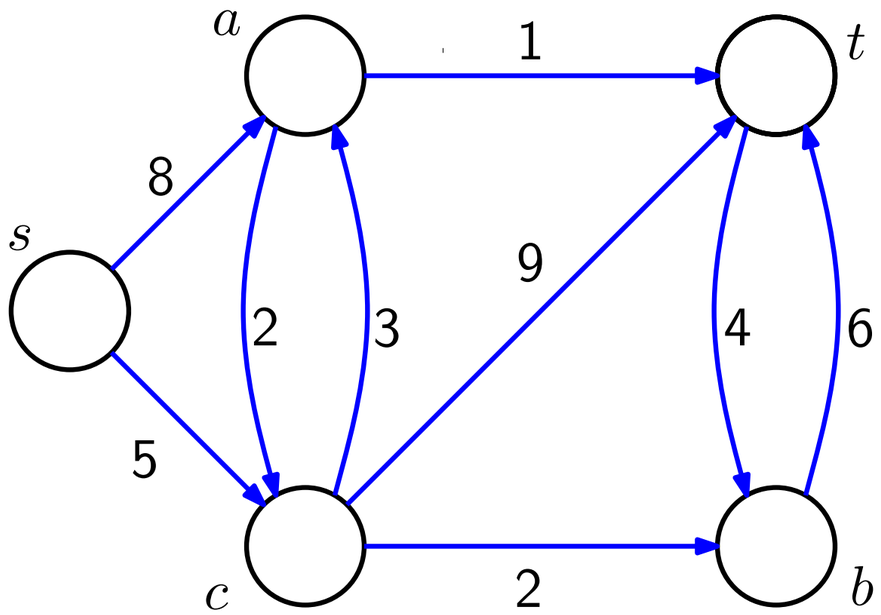



Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$
INITIALIZE$(G, s)$

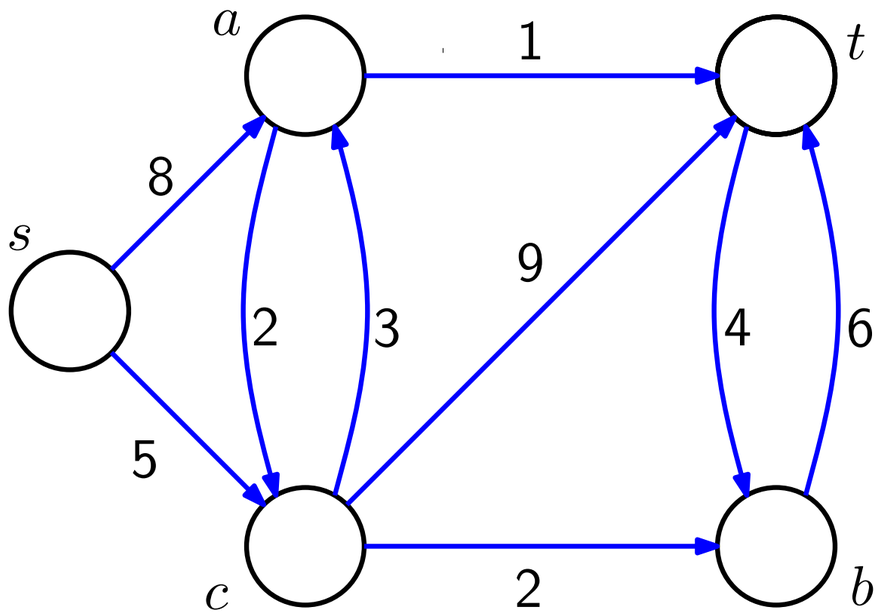



Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$
INITIALIZE$(G, s)$




**function** INITIALIZE$(G, s)$
  **for** $u \in V$ **do**
    $u$.color $=$ white
    $u$.d $= \infty$
    $u.\pi =$ None
  **end for**
  $s$.color $=$ grey
  $s$.d $= 0$
**end function**

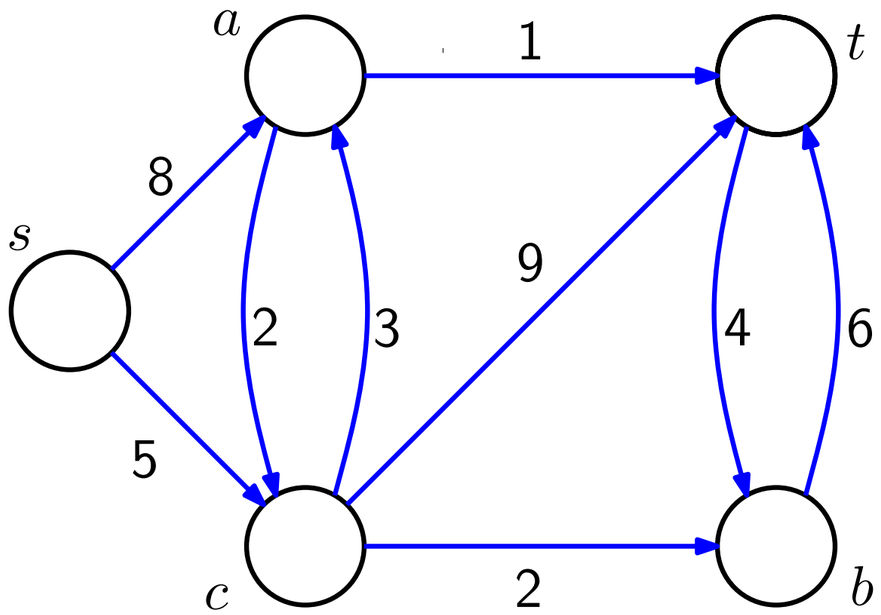
Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$
  INITIALIZE$(G, s)$
  Q = new PriorityQueue$(V, d)$
  **while** not Q.Empty() **do**

  **end while**




**function** INITIALIZE$(G, s)$
  **for** $u \in V$ **do**
    $u$.color = white
    $u$.d = $\infty$
    $u.\pi$ = None
  **end for**
  $s$.color = grey
  $s$.d = 0
**end function**

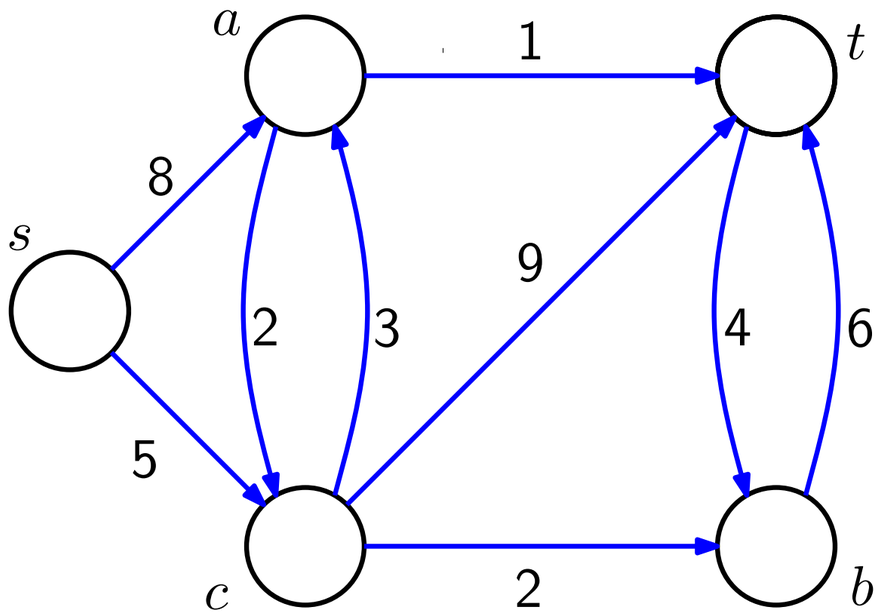
Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$
  INITIALIZE$(G, s)$
  Q = new PriorityQueue$(V, d)$
  **while** not Q.Empty() **do**
    $u$ = Q.ExtractMin()

  **end while**




**function** INITIALIZE$(G, s)$
  **for** $u \in V$ **do**
    $u$.color = white
    $u$.d = $\infty$
    $u.\pi$ = None
  **end for**
  $s$.color = grey
  $s$.d = 0
**end function**

Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

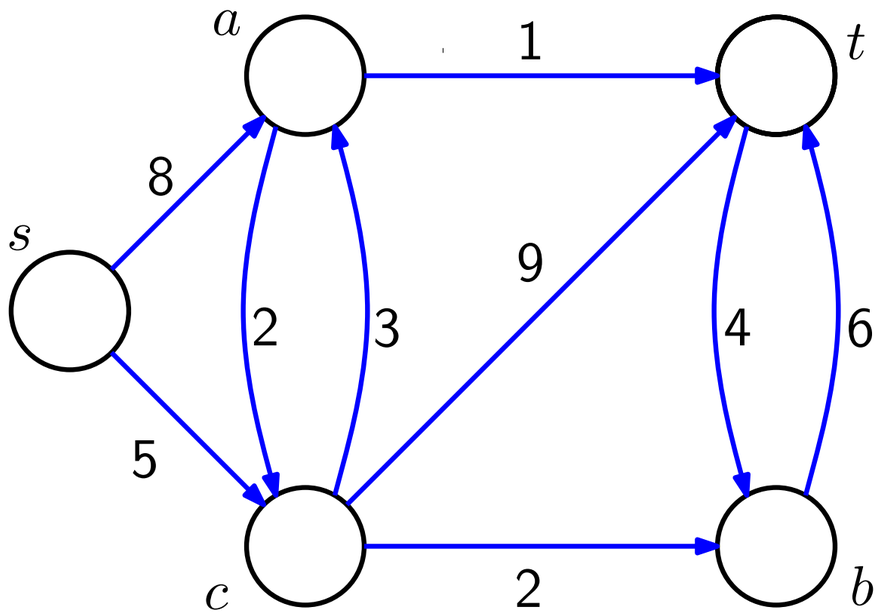
**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$
  INITIALIZE$(G, s)$
  Q = new PriorityQueue$(V, d)$
  **while** not Q.Empty() **do**
    $u$ = Q.ExtractMin()
    **for** $v \in$ Adj$[u]$ **do**

    **end for**

  **end while**




**function** INITIALIZE$(G, s)$
  **for** $u \in V$ **do**
    $u$.color = white
    $u$.d = $\infty$
    $u.\pi$ = None
  **end for**
  $s$.color = grey
  $s$.d = 0
**end function**

Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

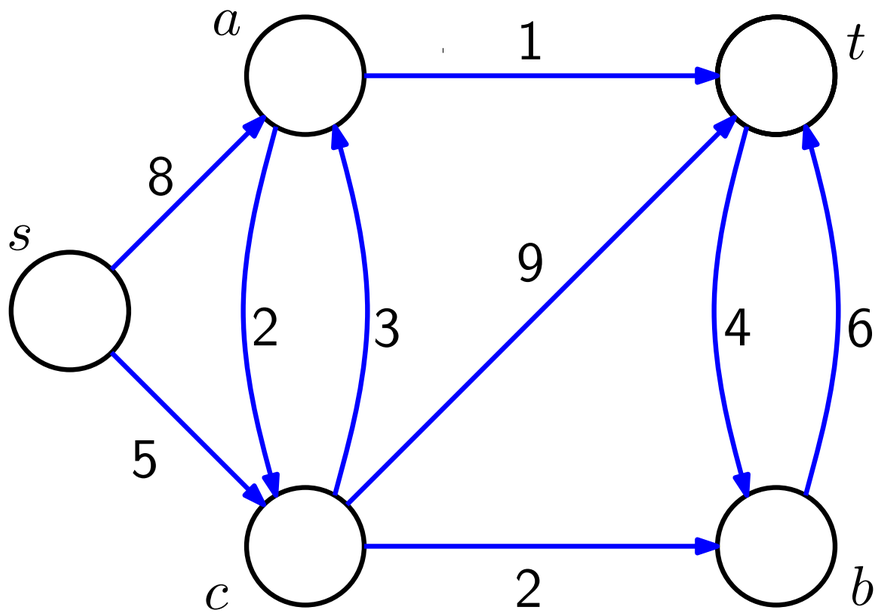
**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$

```
INITIALIZE(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    for v ∈ Adj[u] do
        RELAX(u, v; w)
    end for

end while
```




```
function RELAX(u, v; w)
    if v.d > u.d + w(u, v) then
        v.color = grey
        v.d = u.d + w(u, v)
        v.π = u
        Q.DecreaseKey(v, v.d)
    end if
end function
```

```
function INITIALIZE(G, s)
    for u ∈ V do
        u.color = white
        u.d = ∞
        u.π = None
    end for
    s.color = grey
    s.d = 0
end function
```

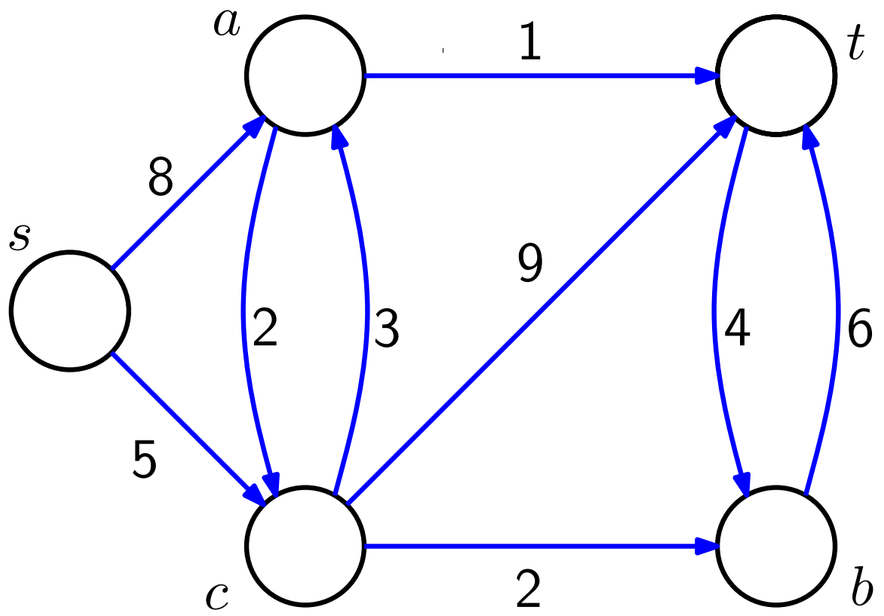
Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Dijkstra

**Input:** (positively) Weighted Graph $G$, Vertex $s$
**Output:** Shortest Paths from $s$

```
INITIALIZE(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    for v ∈ Adj[u] do
        RELAX(u, v; w)
    end for
    u.color = black
end while
```




```
function RELAX(u, v; w)
    if v.d > u.d + w(u, v) then
        v.color = grey
        v.d = u.d + w(u, v)
        v.π = u
        Q.DecreaseKey(v, v.d)
    end if
end function
```

```
function INITIALIZE(G, s)
    for u ∈ V do
        u.color = white
        u.d = ∞
        u.π = None
    end for
    s.color = grey
    s.d = 0
end function
```

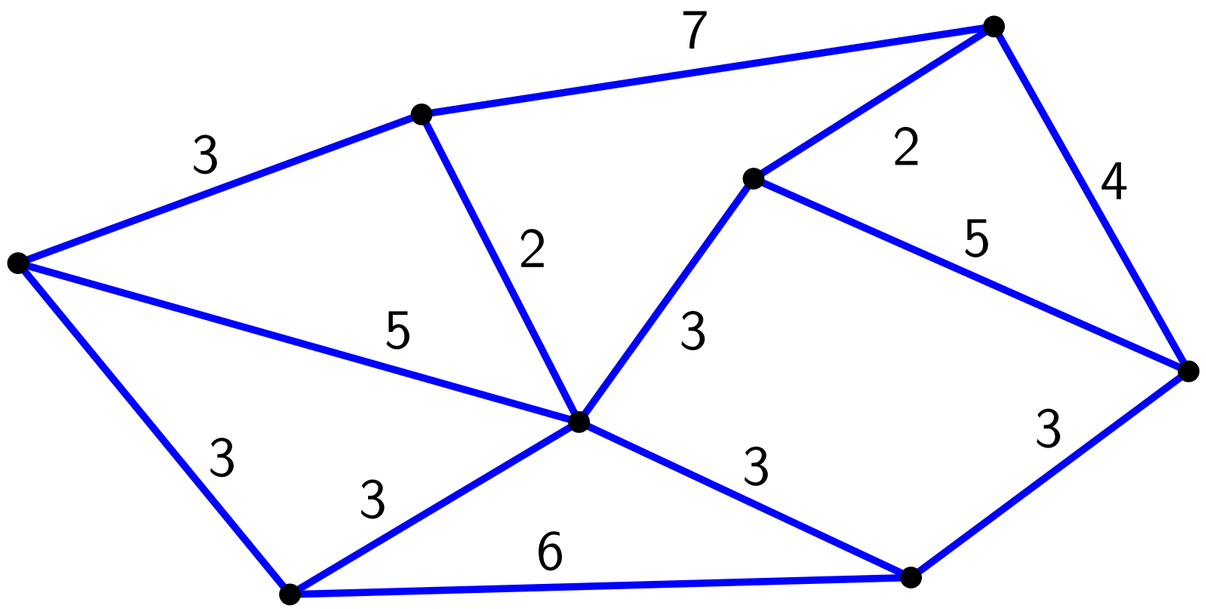
Suche durch iterative Erweiterung um die kürzesten Pfade (greedy), ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Minimale Spannbäume

**Gegeben:**

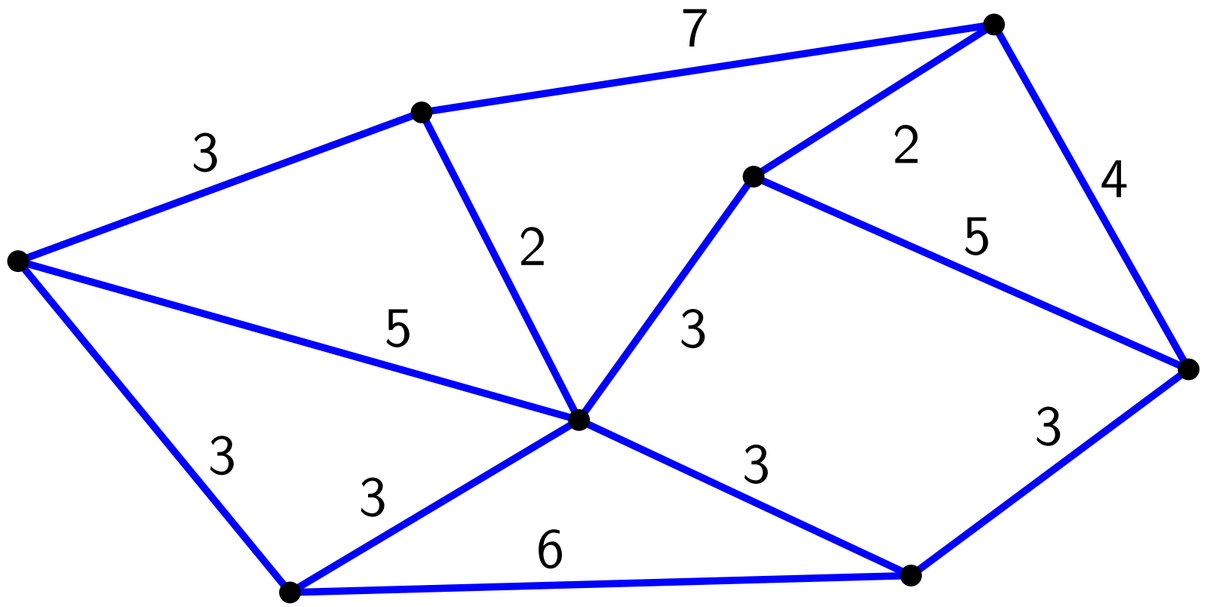
**Gesucht:**



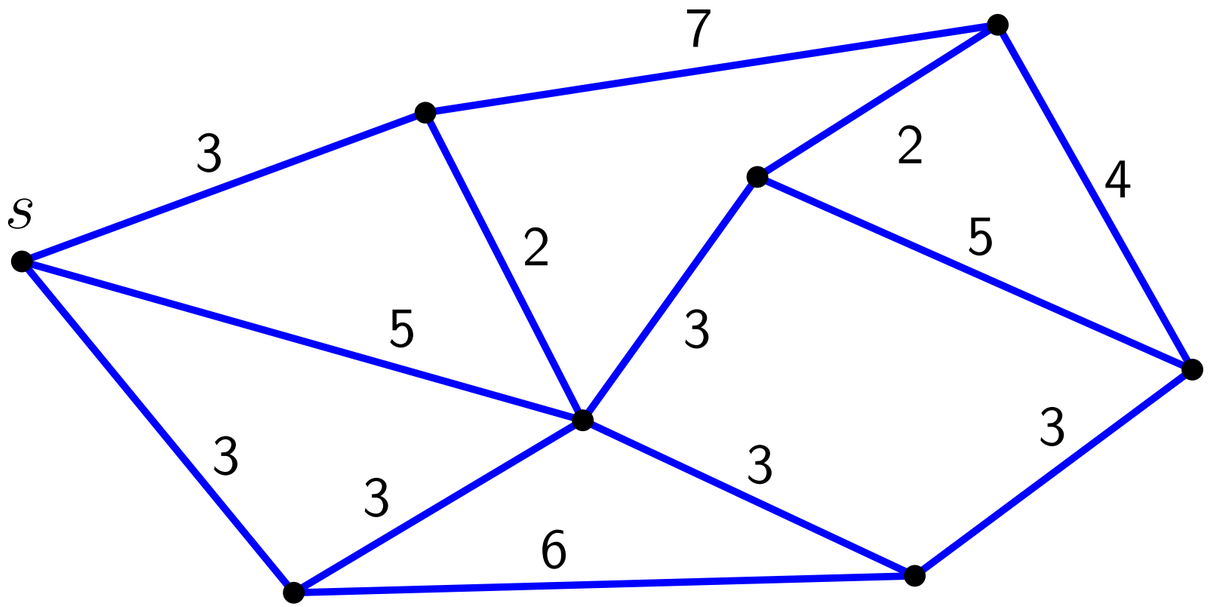
# Minimale Spannbäume

**Gegeben:** Zusammenhängender, ungerichteter Graphen $G = (V, E)$ mit Kantengewichten $w : E \to \mathbb{R}_{>0}$

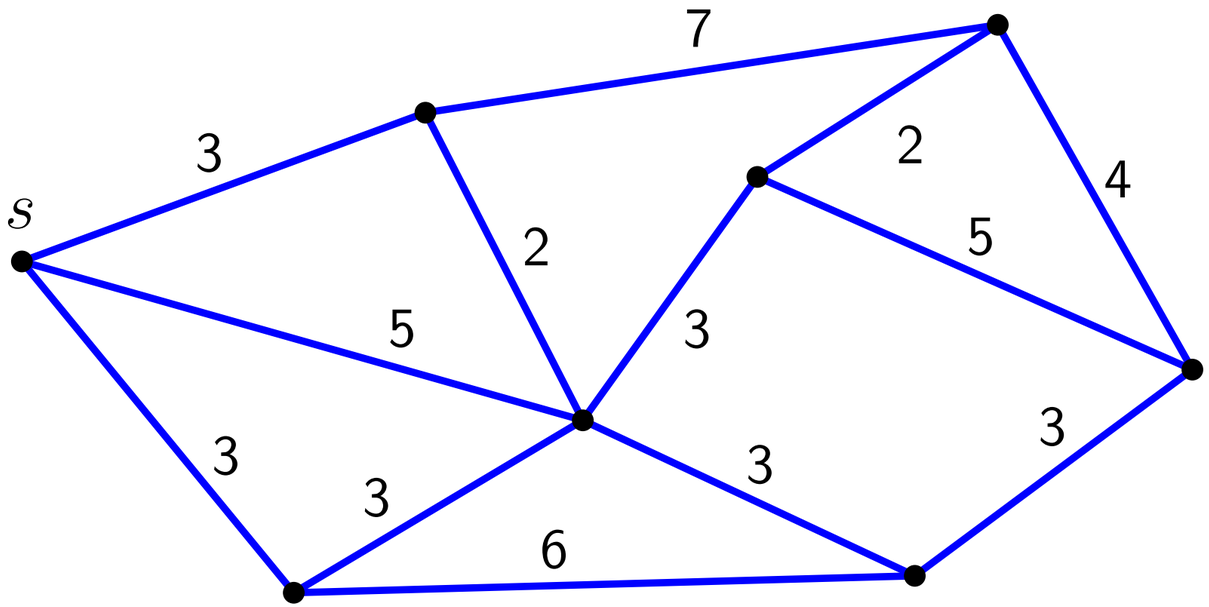
**Gesucht:** Spannbaum $T \subseteq E$ mit minimalen Gesammtgewichten $w(T) = \sum_{e \in T} w(e)$



# Jarník-Prim

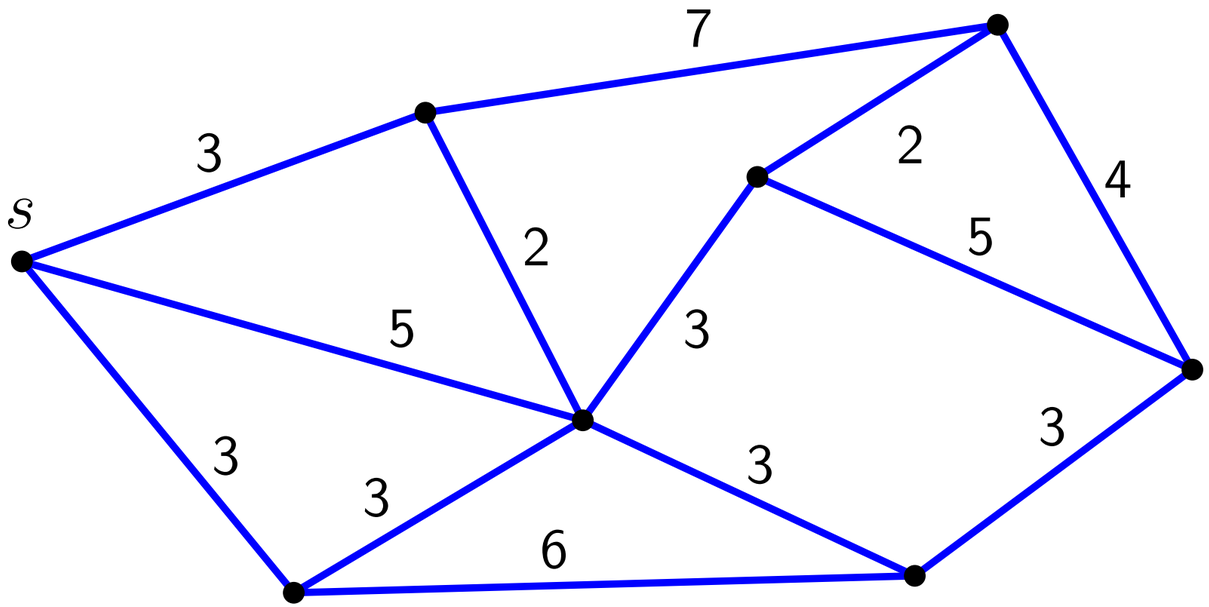


# Jarník-Prim




Suche durch iterative Erweiterung um die günstigsten Kanten, ausgehend vom Startknoten.

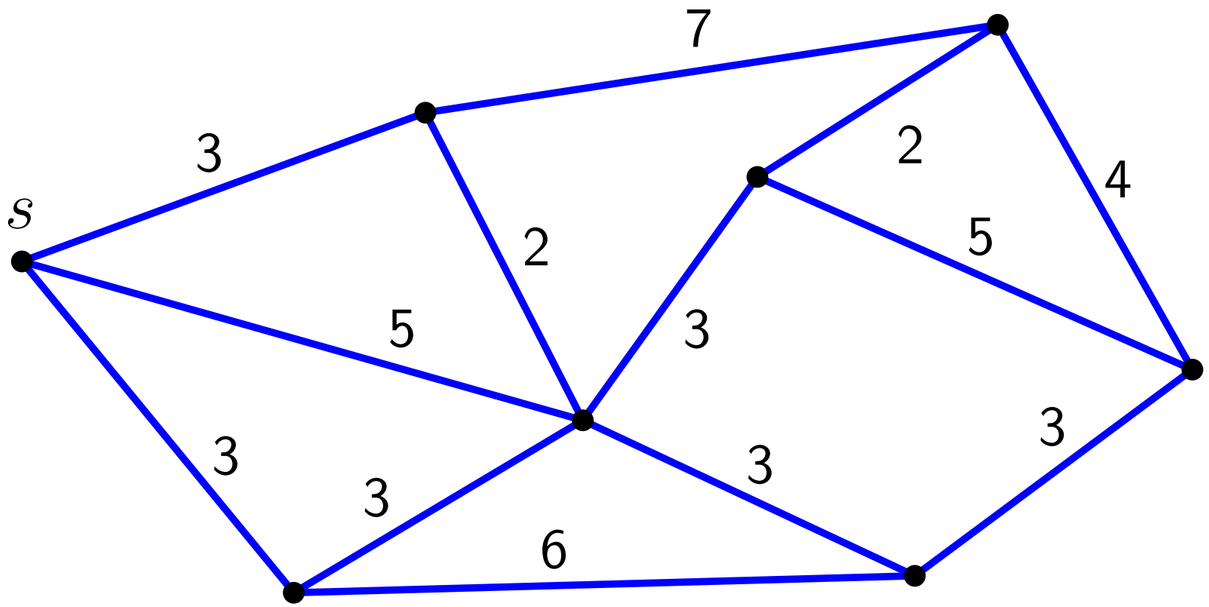
# Jarník-Prim




Suche durch iterative Erweiterung um die günstigsten Kanten, ausgehend vom Startknoten.

**Laufzeit**:

# Jarník-Prim

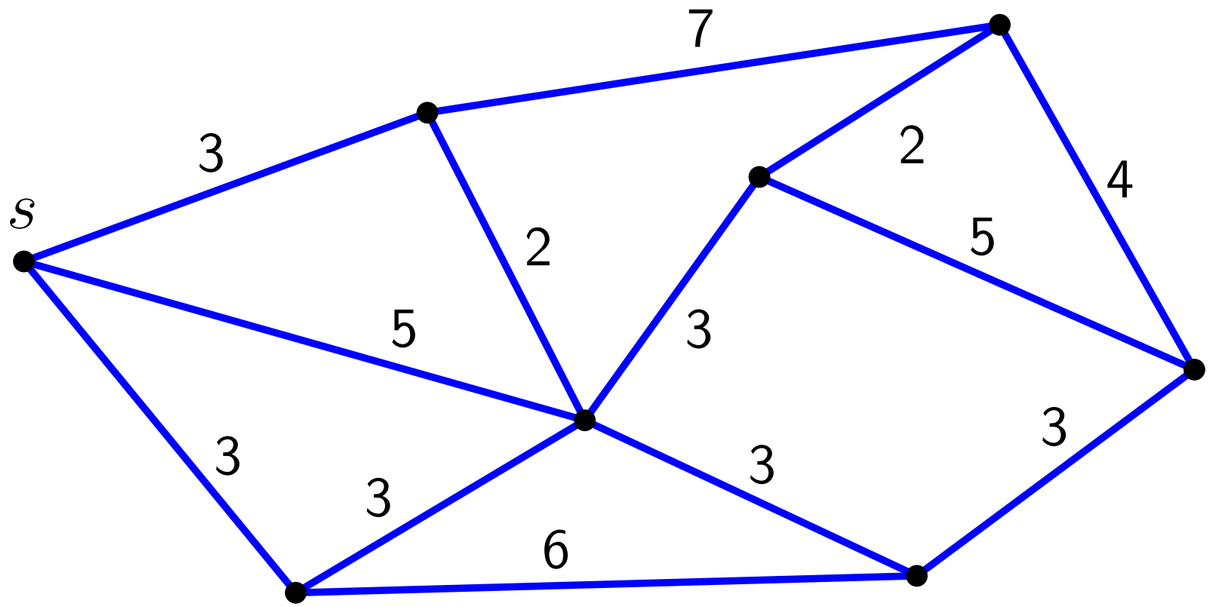



Suche durch iterative Erweiterung
um die günstigsten Kanten,
ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Jarník-Prim

**Input:**
**Output:**

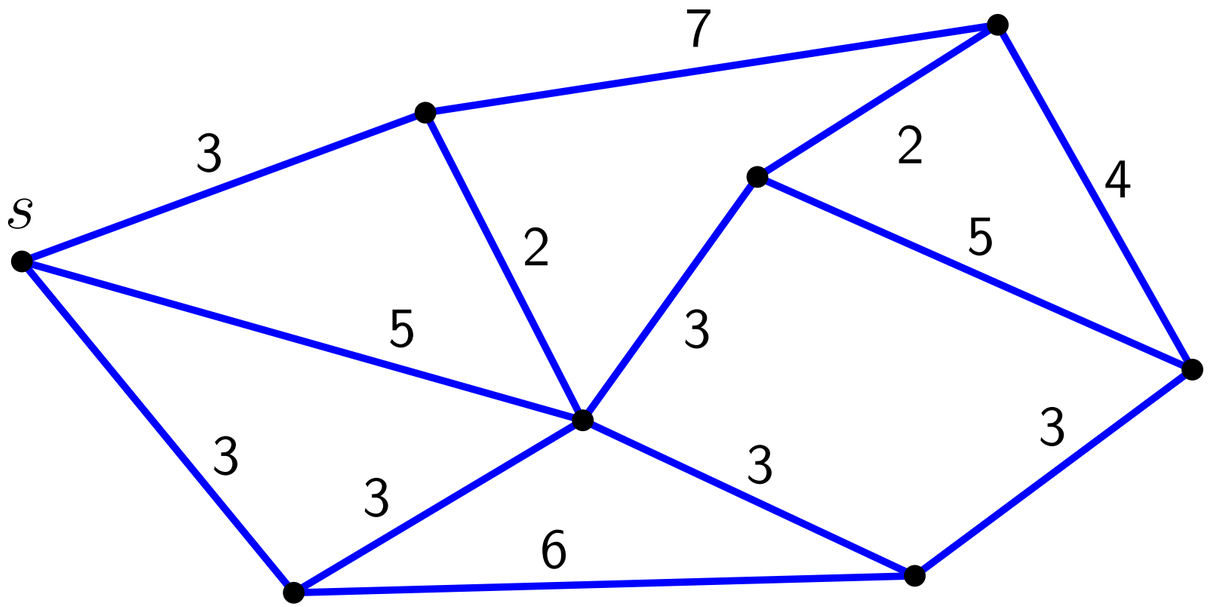



Suche durch iterative Erweiterung um die günstigsten Kanten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Jarník-Prim

**Input:** Weighted Graph $G$, Vertex $s$
**Output:** Minimum Spanning Tree

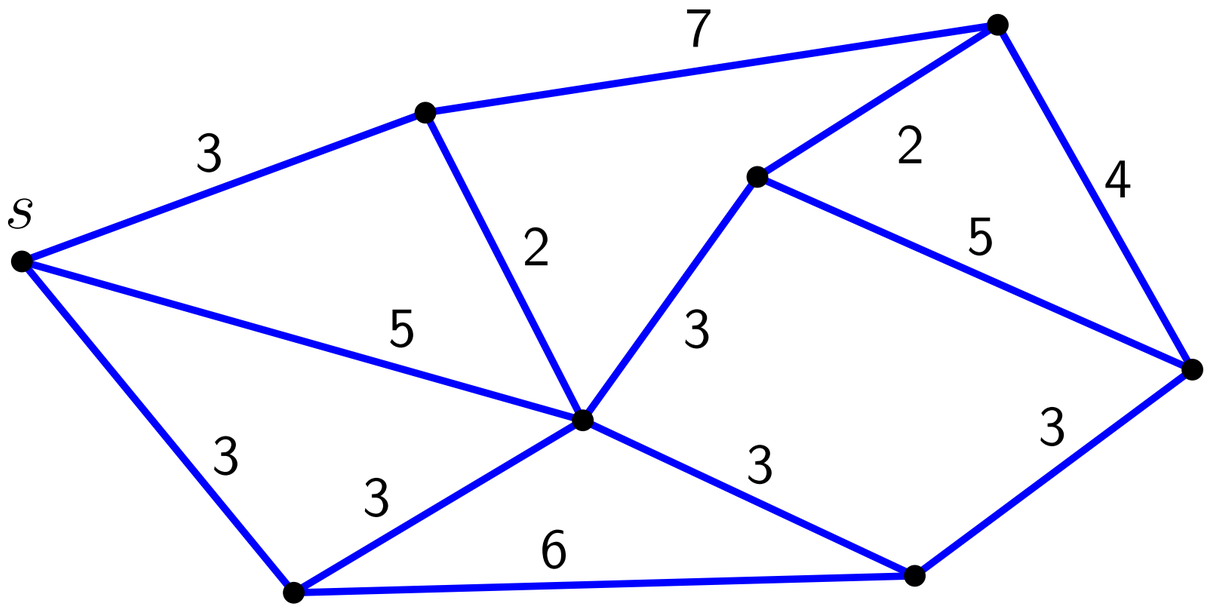



Suche durch iterative Erweiterung
um die günstigsten Kanten,
ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Jarník-Prim

**Input:** Weighted Graph $G$, Vertex $s$
**Output:** Minimum Spanning Tree
  INITIALIZE$(G, s)$
  Q = new PriorityQueue$(V, d)$




**function** INITIALIZE$(G, V)$
  **for** $u \in V$ **do**
    $u$.d $= \infty$
    $u.\pi =$ None
  **end for**
**end function**

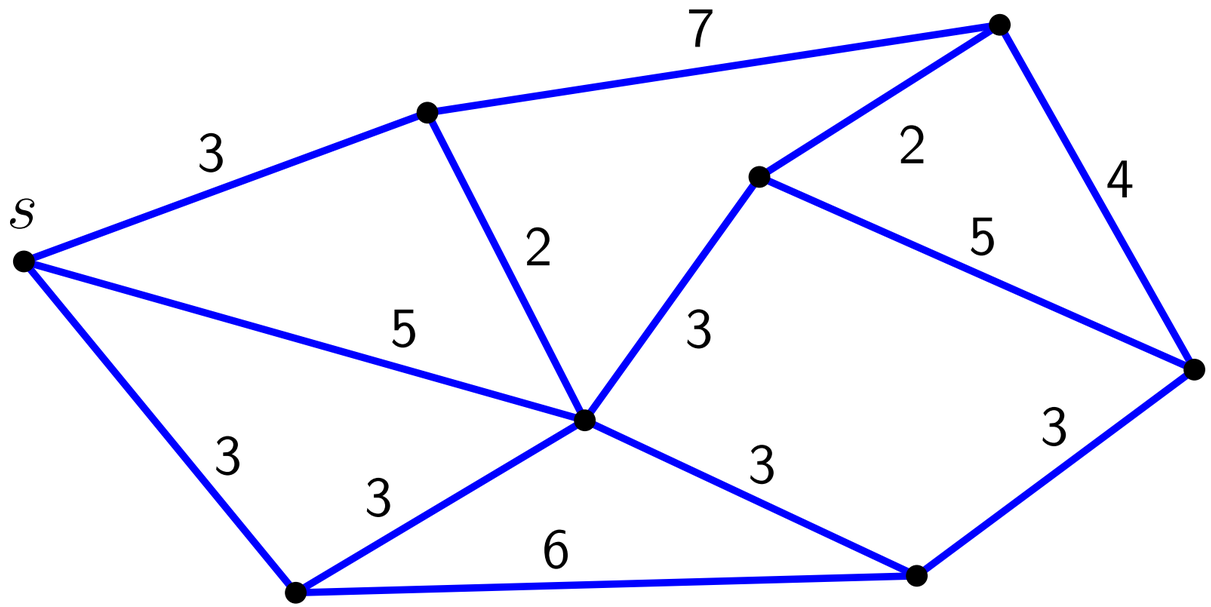
Suche durch iterative Erweiterung um die günstigsten Kanten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Jarník-Prim

**Input:** Weighted Graph $G$, Vertex $s$
**Output:** Minimum Spanning Tree
  INITIALIZE$(G, s)$
  Q = new PriorityQueue$(V, d)$
  **while** not Q.Empty() **do**
    $u$ = Q.ExtractMin()

  **end while**




**function** INITIALIZE$(G, V)$
  **for** $u \in V$ **do**
    $u$.d $= \infty$
    $u.\pi$ = None
  **end for**
**end function**

Suche durch iterative Erweiterung um die günstigsten Kanten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

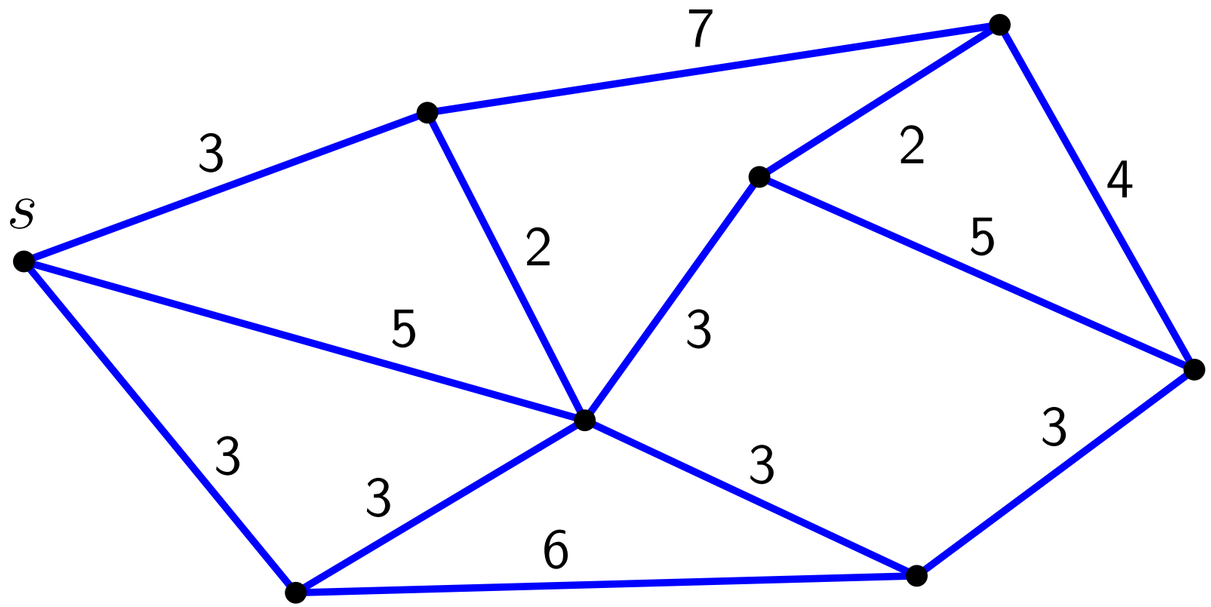
# Jarník-Prim

**Input:** Weighted Graph $G$, Vertex $s$
**Output:** Minimum Spanning Tree

```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    for v ∈ Adj[u] do
        Relax(u, v; w)
    end for
end while
```

```
function Relax(u, v; w)
    if v ∈ Q and v.d > w(u, v) then
        v.d = w(u, v)
        v.π = u
        Q.UpdateKey(v, v.d)
    end if
end function
```

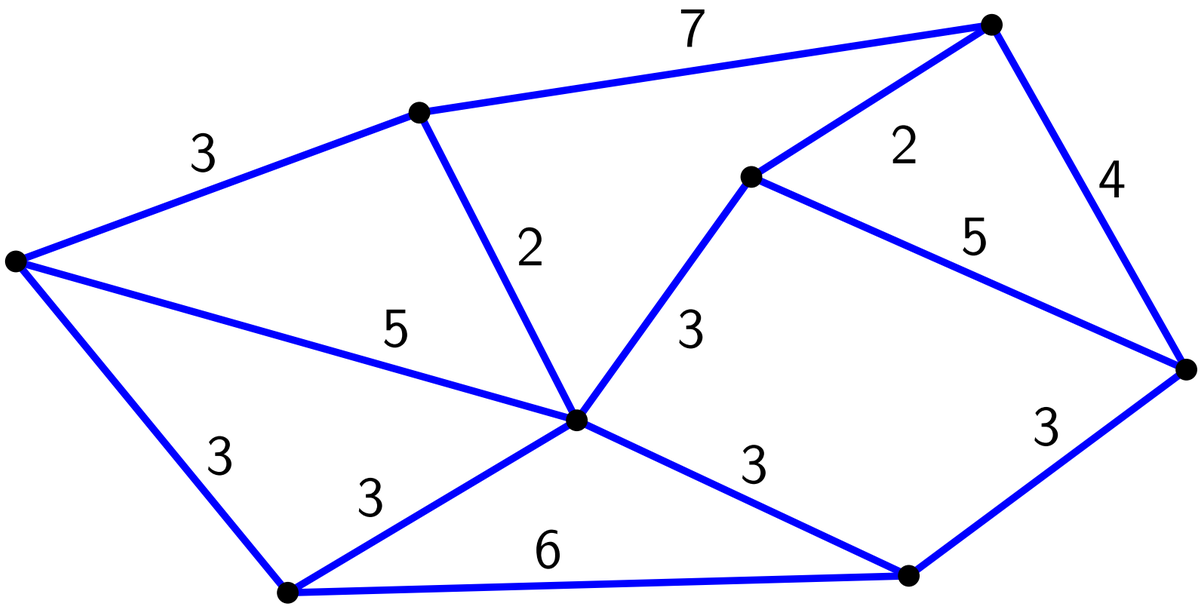
```
function Initialize(G, V)
    for u ∈ V do
        u.d = ∞
        u.π = None
    end for
end function
```



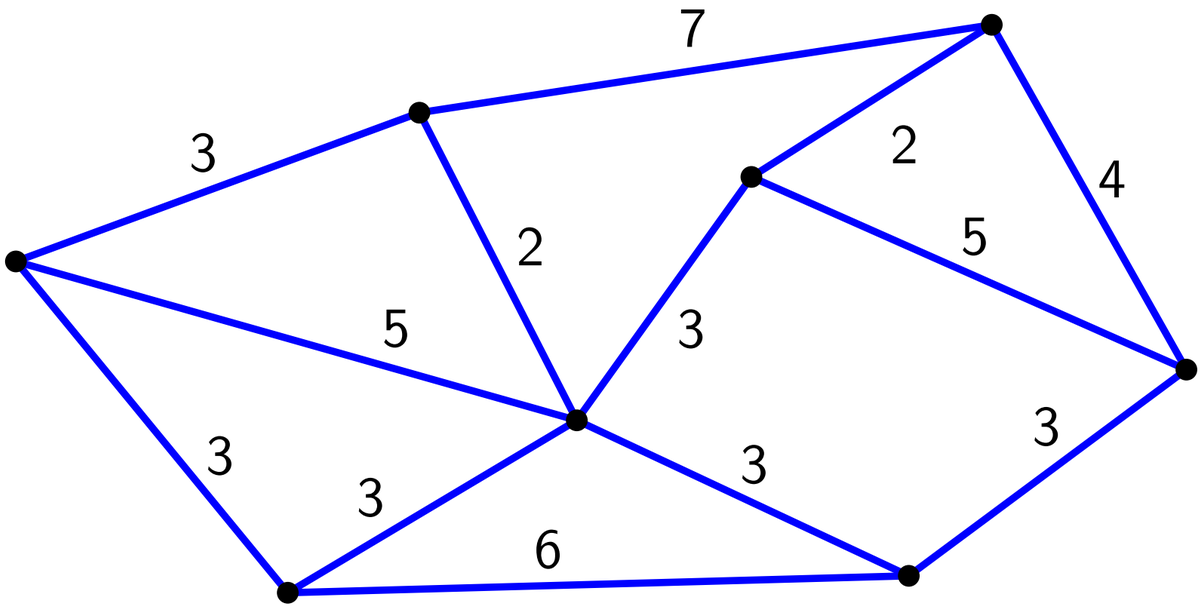

Suche durch iterative Erweiterung um die günstigsten Kanten, ausgehend vom Startknoten.

**Laufzeit**: $O(|V| \cdot \log(|V|) + |E|)$

# Kruskal

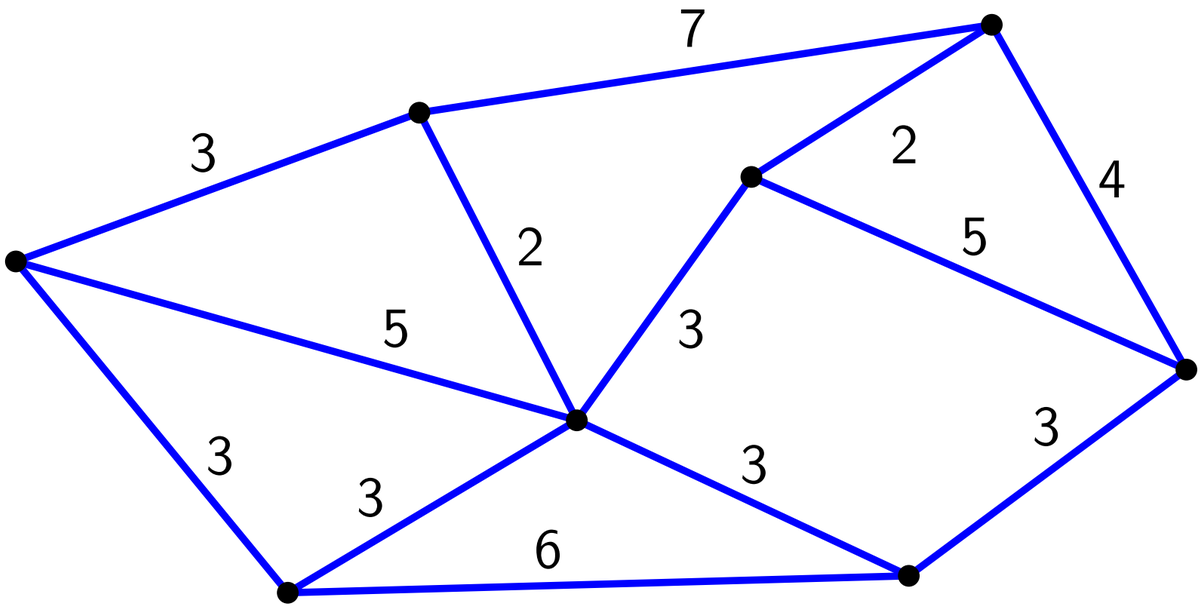


# Kruskal




Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

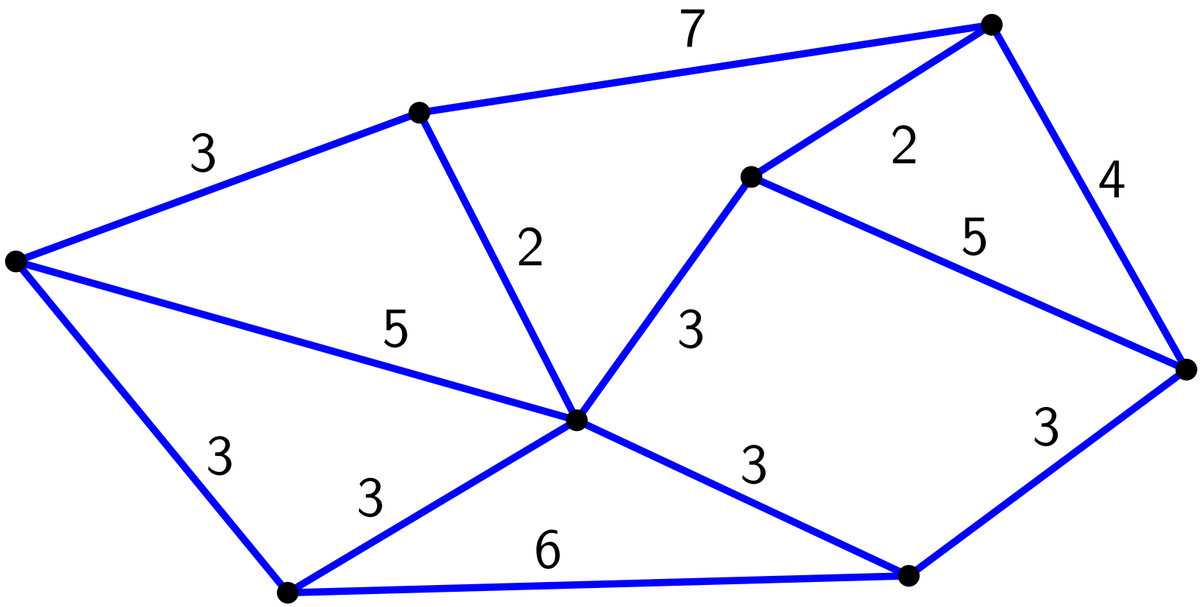
# Kruskal




Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**:

# Kruskal



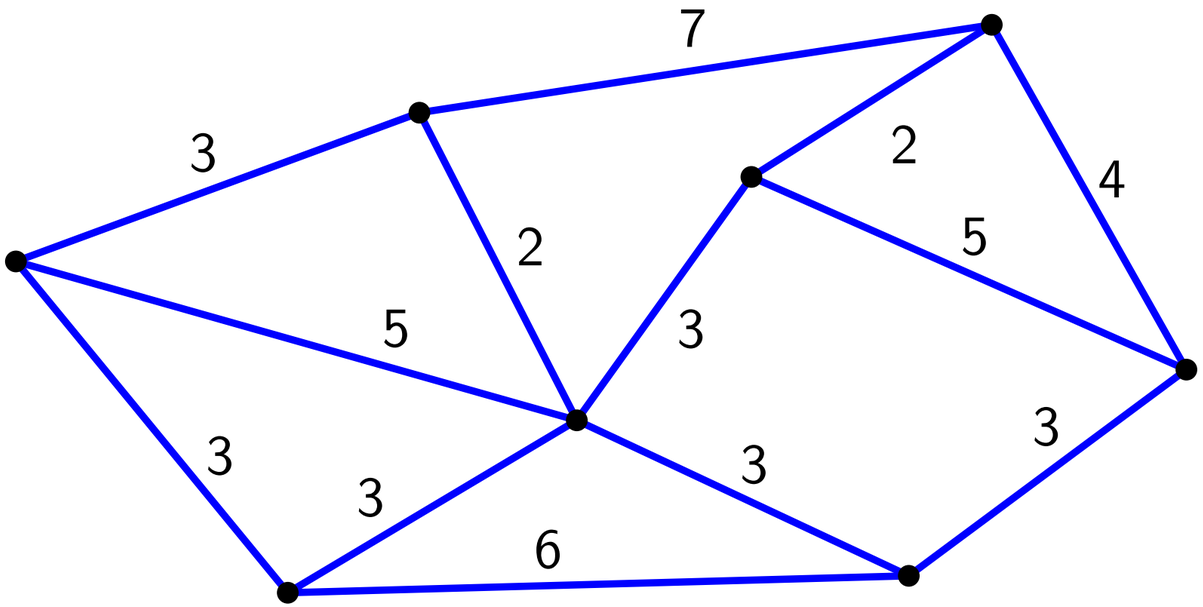

Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**: $O(|E| \cdot \log(|V|))$

# Kruskal

**Input:**

**Output:**



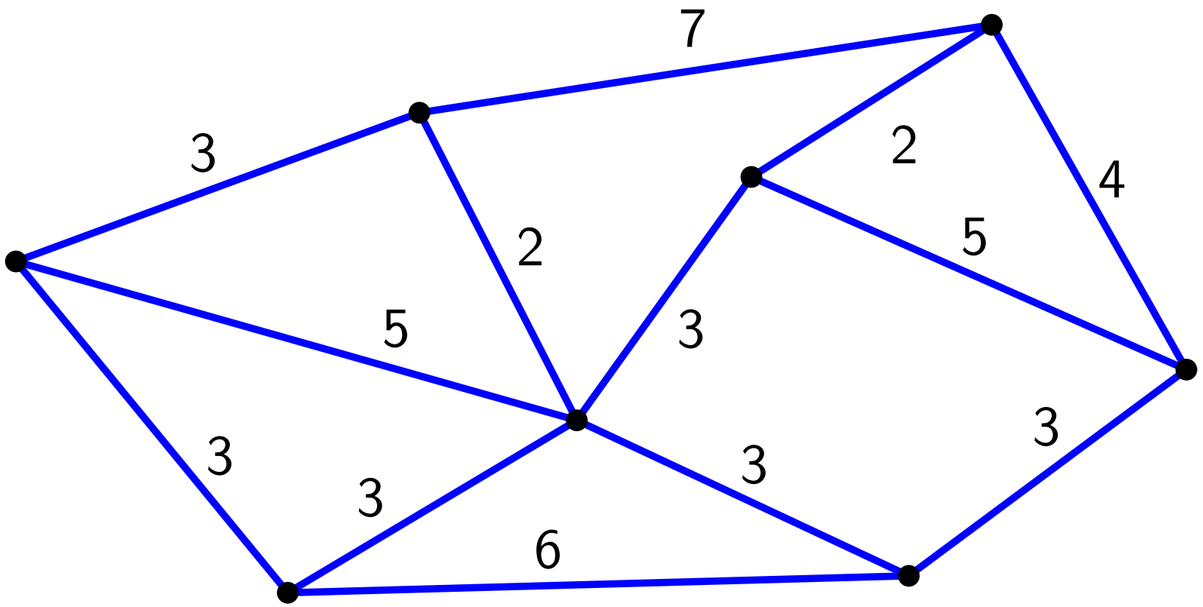

Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**: $O(|E| \cdot \log(|V|))$

# Kruskal

**Input:** Weighted, undirected Graph $G = (V, E; w)$

**Output:** Minimum weight forest $T$

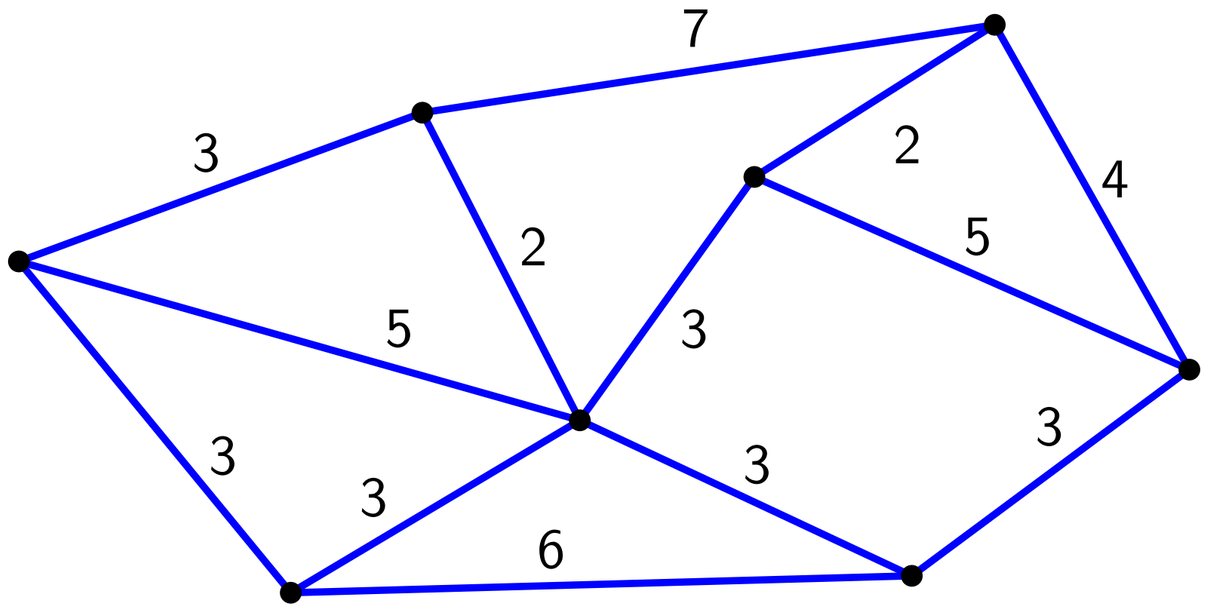



Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**: $O(|E| \cdot \log(|V|))$

# Kruskal

**Input:** Weighted, undirected Graph $G = (V, E; w)$
**Output:** Minimum weight forest $T$
$T = \emptyset$
**for** $v \in V$ **do**
    MakeSet($v$)
**end for**

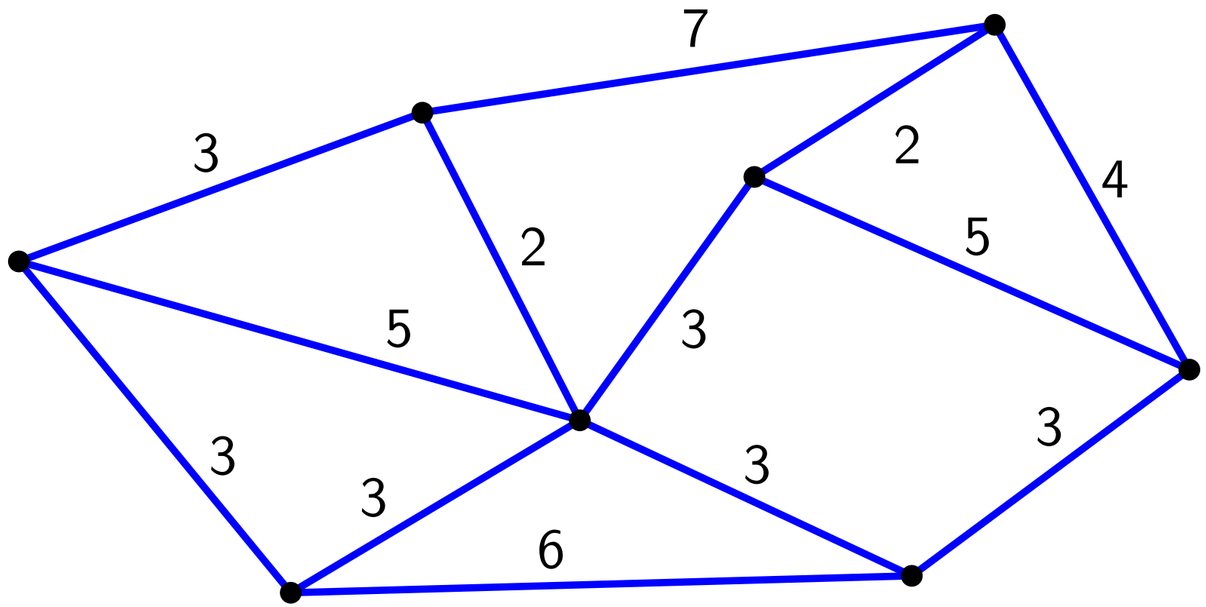



Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**: $O(|E| \cdot \log(|V|))$

# Kruskal

**Input:** Weighted, undirected Graph $G = (V, E; w)$
**Output:** Minimum weight forest $T$
$T = \emptyset$
**for** $v \in V$ **do**
    MakeSet($v$)
**end for**
Sort $E$ ascending by weights $w$



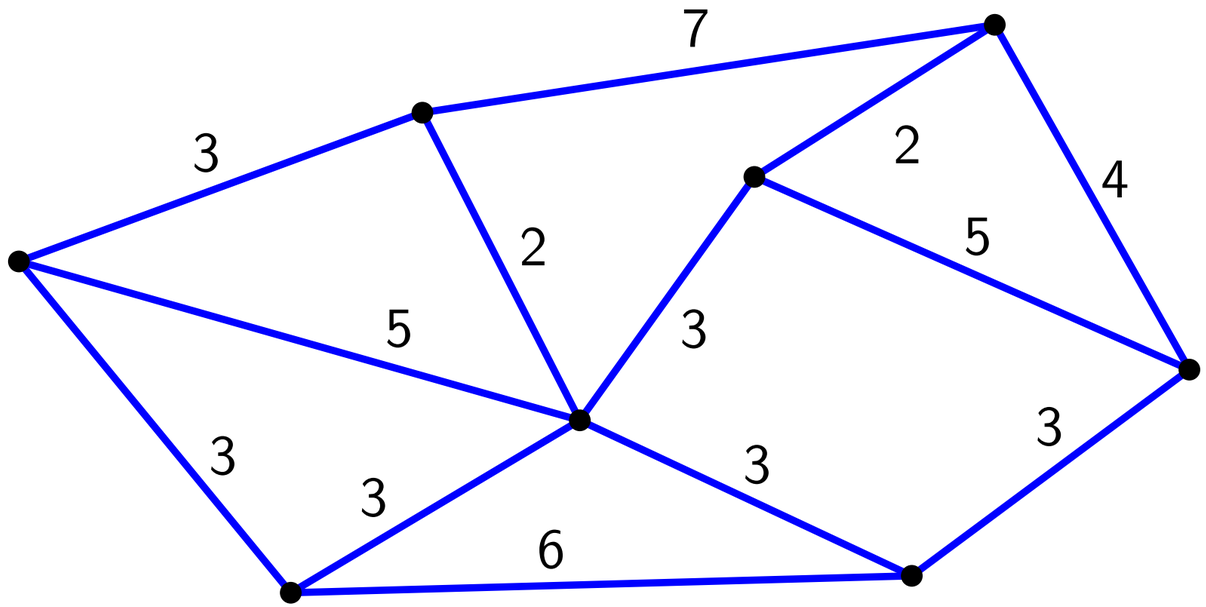

Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**: $O(|E| \cdot \log(|V|))$

# Kruskal

**Input:** Weighted, undirected Graph $G = (V, E; w)$
**Output:** Minimum weight forest $T$
$T = \emptyset$
**for** $v \in V$ **do**
    MakeSet($v$)
**end for**
Sort $E$ ascending by weights $w$
**for** $(u, v) \in E$ **do**
    **if** FindSet($u$) $\neq$ FindSet($v$) **then**

    **end if**
**end for**



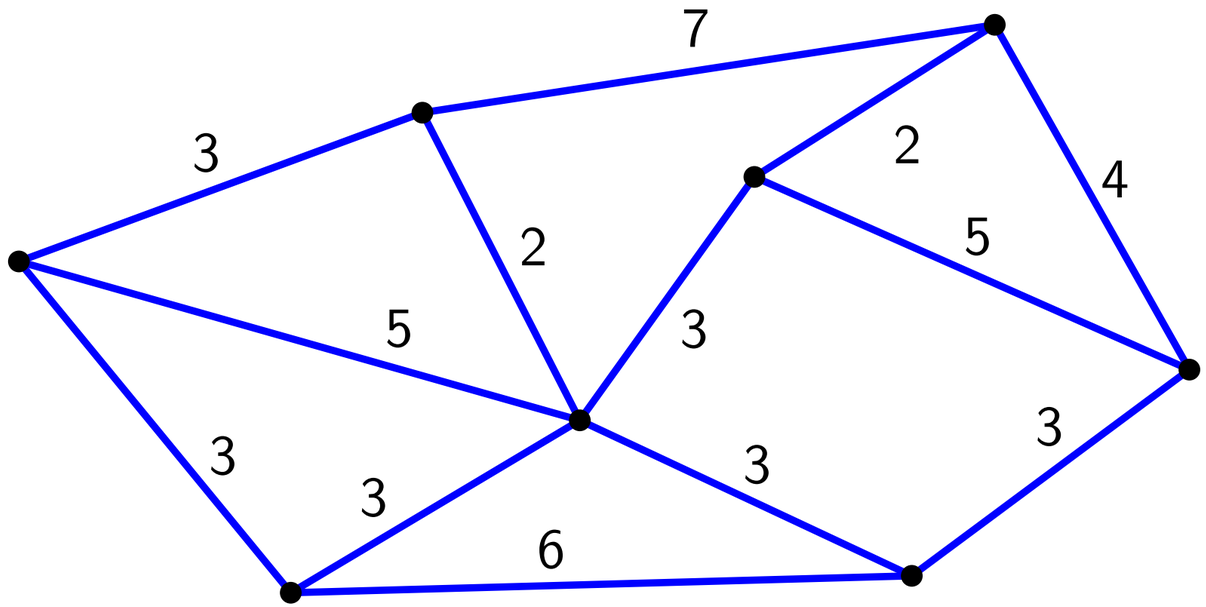

Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**: $O(|E| \cdot \log(|V|))$

# Kruskal

**Input:** Weighted, undirected Graph $G = (V, E; w)$
**Output:** Minimum weight forest $T$

```
T = ∅
for v ∈ V do
    MakeSet(v)
end for
Sort E ascending by weights w
for (u, v) ∈ E do
    if FindSet(u) ≠ FindSet(v) then
        T = T ∪ (u, v)
        Union(u, v)
    end if
end for
```




Greedy Suche durch iterative Erweiterung um die günstigsten Kanten bis ein Zyklus gefunden wird.

**Laufzeit**: $O(|E| \cdot \log(|V|))$

# Viel Erfolg beim ersten Übungsblatt!

Fragen? Anregungen?