Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

Wiederholung:

Grundlegende Algorithmen aus der Vorlesung ADS

Alexander Wolff

Sommersemester 2021

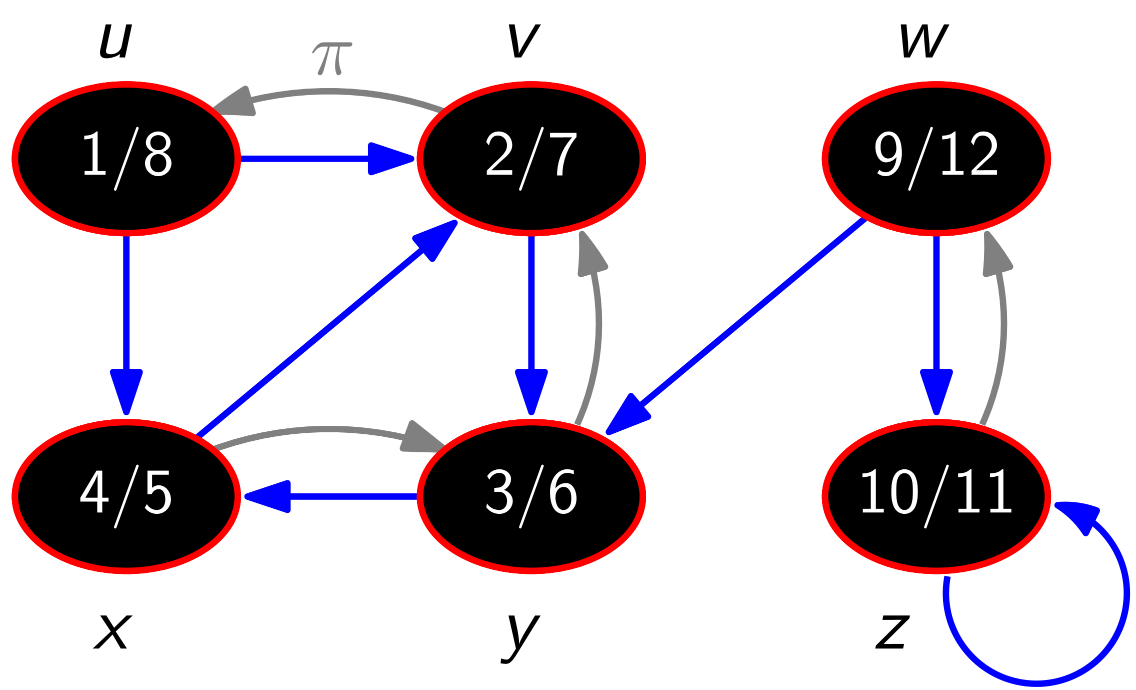
Übersicht

Tiefensuche – Beispiel

Eingabe: (un)gerichteter Graph G

Ausgabe: – Besuchsintervalle $(\underbrace{u.d}_{\text{discovery time}} / \underbrace{u.f}_{\text{finish time}})$

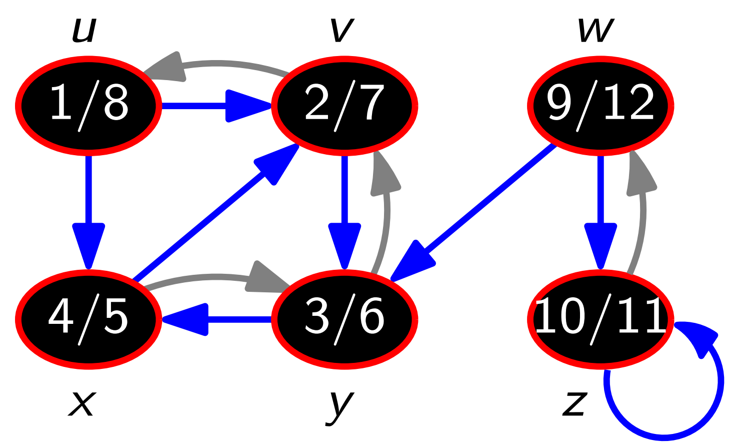
– DFS-Wald $\left(\xleftarrow{\pi}\right)$



Tiefensuche – Pseudocode

```
DFS(Graph G = (V, E))
  foreach u ∈ V do
    u.color = white
    u.π = nil
  time = 0      // globale Variable!
  foreach u ∈ V do
    if u.color == white then DFSVisit(G, u)
```

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



Laufzeit?

- DFSVisit wird nur für weiße Knoten aufgerufen.
- In DFSVisit wird der neue Knoten sofort gefärbt.

$\Rightarrow$ DFSVisit wird für jeden Knoten genau $1\times$ aufgerufen.

- Jede Kante wird *insgesamt* höchstens $2\times$ betrachtet.

DFS gesamt $\quad O(V + E)$ Zeit

Tiefensuche – Anwendung

Topologische Sortierung: Lineare Ordnung der Knoten, so dass aus $(u, v) \in E$ folgt: u kommt vor v.

$\Rightarrow$ kreisfrei

TopologicalSort(DirectedGraph G)
  L = **new** List()
  DFS(G) mit folgender Änderung:
    Wenn ein Knoten schwarz gefärbt wird,
    häng ihn *vorne* an die Liste L an.
  return L

Laufzeit?
$O(V + E)$

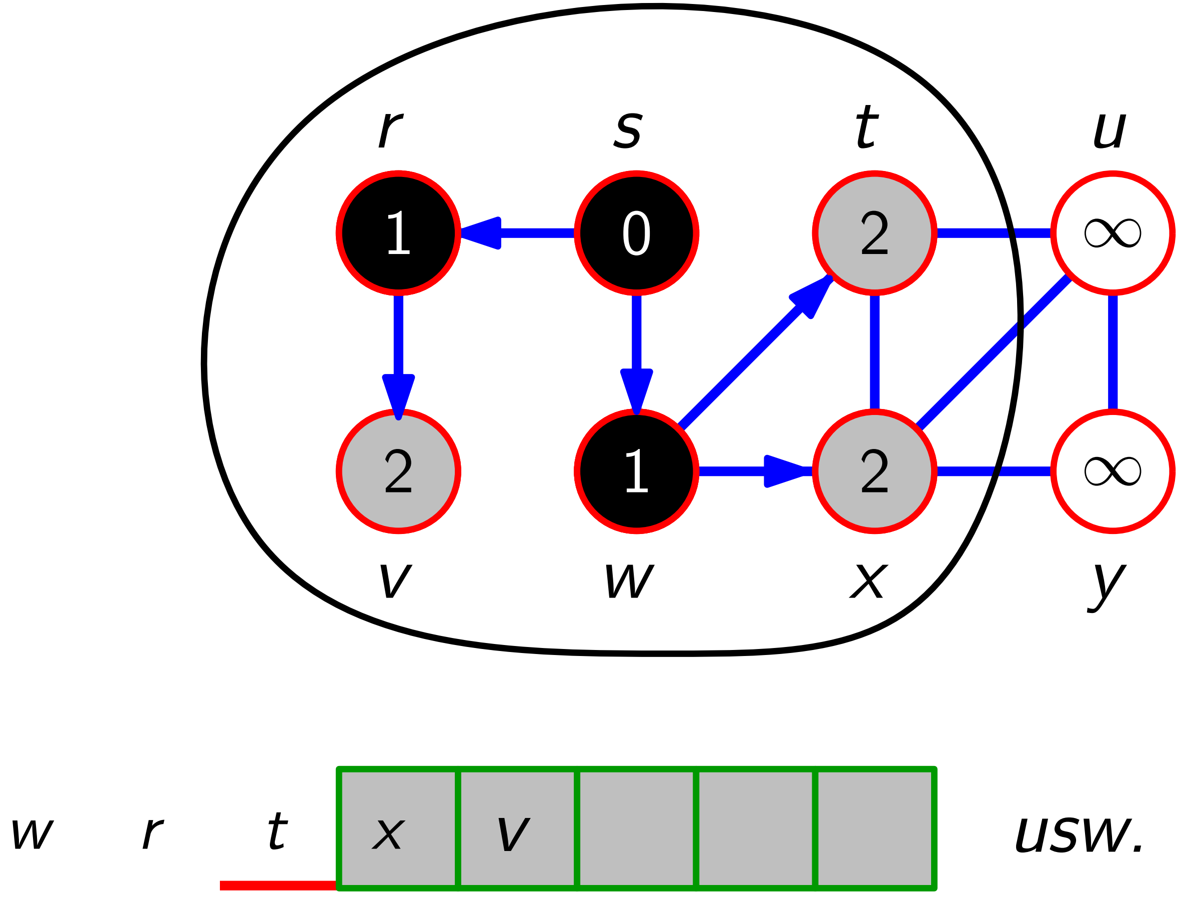
Einen Graphen topologisch zu sortieren ist ein wichtiger Vorverarbeitungsschritt bei der Lösung vieler Probleme – z.B. in der Ablaufplanung, wo gerichtete Kanten Abhängigkeiten von Aufträgen ausdrücken.

Übersicht

Breitensuche – Beispiel

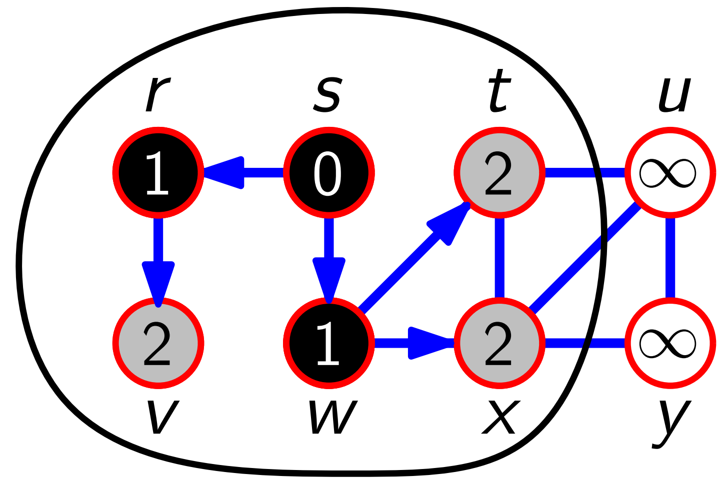
Eingabe: (un)gerichteter Graph G

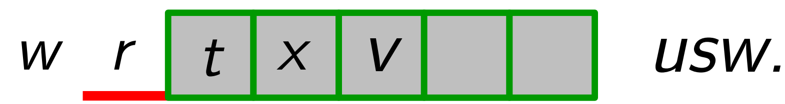
Ausgabe: – Abstand vom Startknoten
– BFS-Wald



Breitensuche – Pseudocode

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



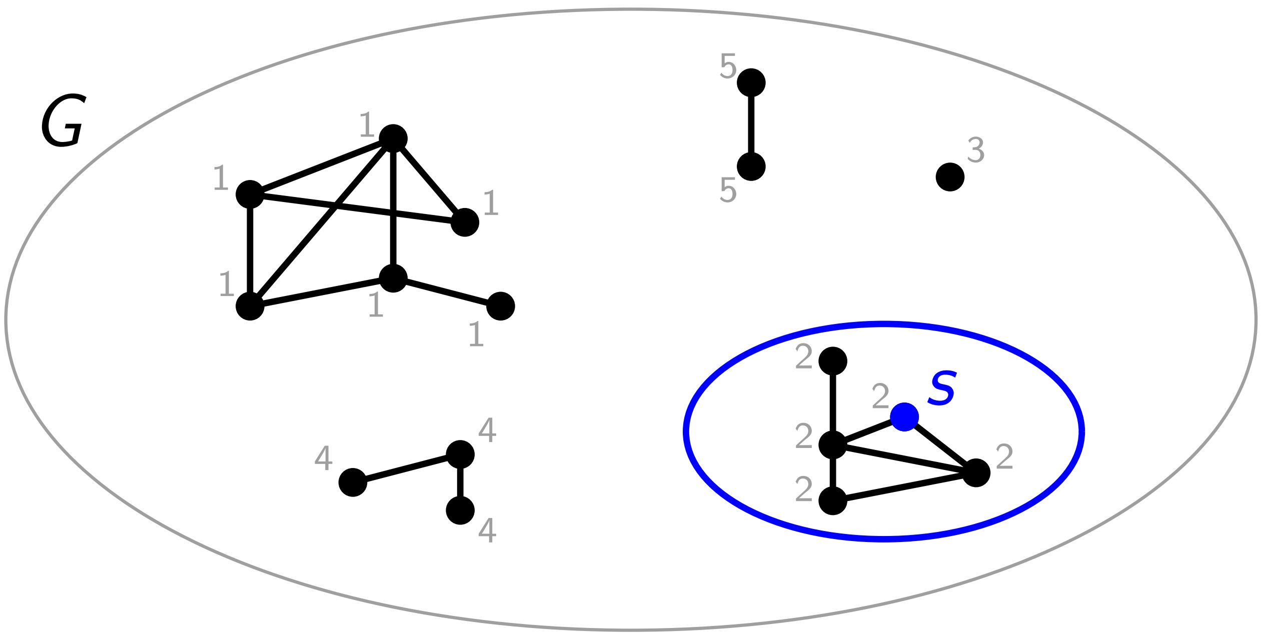



```
Initialize(Graph G, Vertex s)
   foreach u ∈ V do
      u.color = white
      u.d = ∞
      u.π = nil
   s.color = gray
   s.d = 0
```

Laufzeit? $\underbrace{O(V)}_{\text{Initialize}} + \underbrace{O(V)}_{\text{En-/Dequeues}} + \underbrace{O(E)}_{\text{Adjazenzlisten (foreach-Schleifen)}} = O(V+E)$

Breitensuche – Anwendung

Zusammenhangskomponente:
Maximale Teilmenge von Knoten, die über Wege miteinander verbunden sind.



G hat also fünf Zusammenhangskomponenten.

Laufzeit fürs Zählen aller Zusammenhangskomponenten:

Auch (nur) $O(V + E)$!

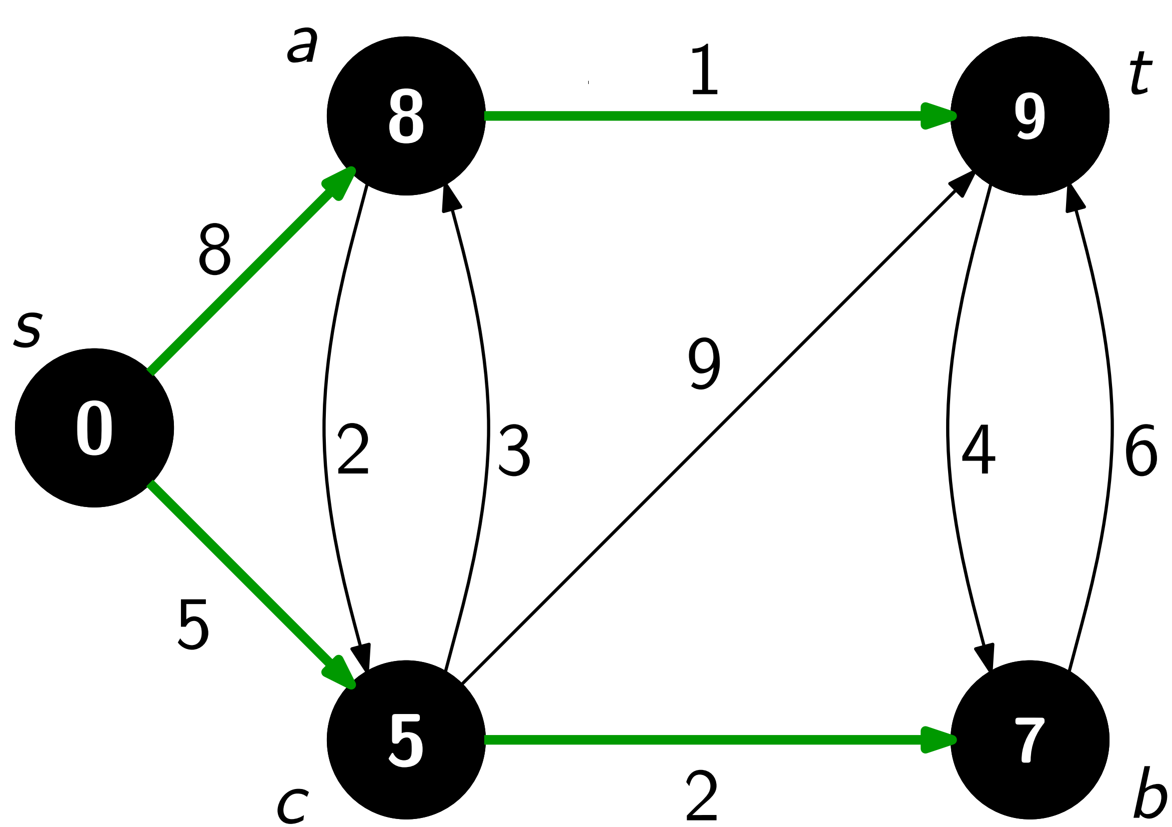
Eine weitere wichtige Anwendung der Breitensuche ist die schnelle Berechnung von kürzesten Wegen in ungewichteten Graphen.

Übersicht

Dijkstra – Beispiel

Eingabe: gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen Kantengewichten und Knoten s

Ausgabe: kürzeste s-t-Wege in G mit Vorgänger-Zeiger π



Dijkstra – Pseudocode

Dijkstra(WeightedGraph G, Vertex s)

  Initialize(G, s)

  $Q =$ **new** PriorityQueue(V, d)

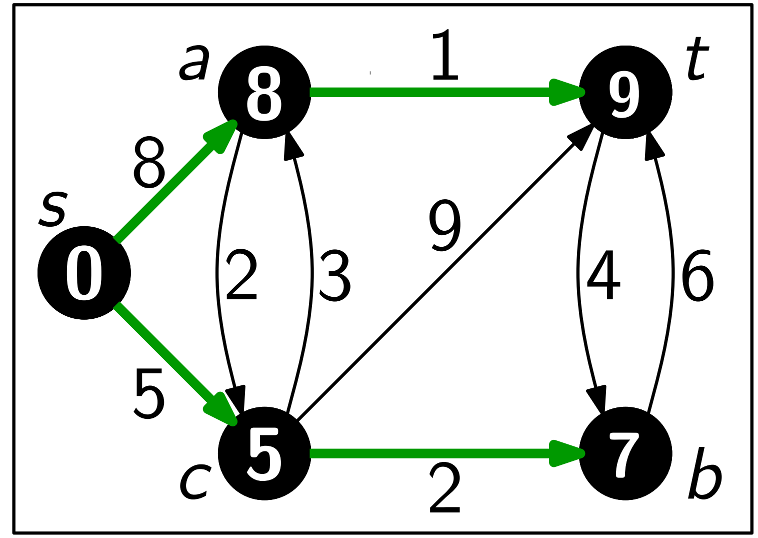
  while not Q.Empty() **do**

    $u = Q$.ExtractMin()

    foreach $v \in \mathrm{Adj}[u]$ **do**

      Relax($u, v; w$)

    $u.color = black$



Relax($u, v; w$)

  if $v.d > u.d + w(u, v)$ **then**

    $v.color = gray$

    $v.d = u.d + w(u, v)$

    $v.\pi = u$

    Q.DecreaseKey($v, v.d$)

Initialize(Graph G, Vertex s)

  foreach $u \in V$ **do**

    $u.color = white$

    $u.d = \infty$

    $u.\pi = nil$

  $s.color = gray$

  $s.d = 0$

Dijkstra – Pseudocode

Laufzeit?

Dijkstra(WeightedGraph G, Vertex s)
  Initialize(G, s)
  Q = **new** PriorityQueue(V, d)
  while not Q.Empty() **do**
    $u = Q$.ExtractMin()
    foreach $v \in \mathrm{Adj}[u]$ **do**
      Relax($u, v; w$)
    $u.color = black$

$O(V)$ Zeit

Genau $|V|$ mal.

Für jeden Knoten $u \in V$ genau $|\mathrm{Adj}[u]|$ ($=\deg u$) mal, also insg. $2|E|$ mal.

Relax($u, v; w$)
  if $v.d > u.d + w(u, v)$ **then**
    $v.color = gray$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
    Q.DecreaseKey($v, v.d$)

Also wird DecreaseKey $\leq 2|E|$ mal aufgerufen.

Gesamt: Implementierungsabh.!
beste: $O(E + V \log V)$

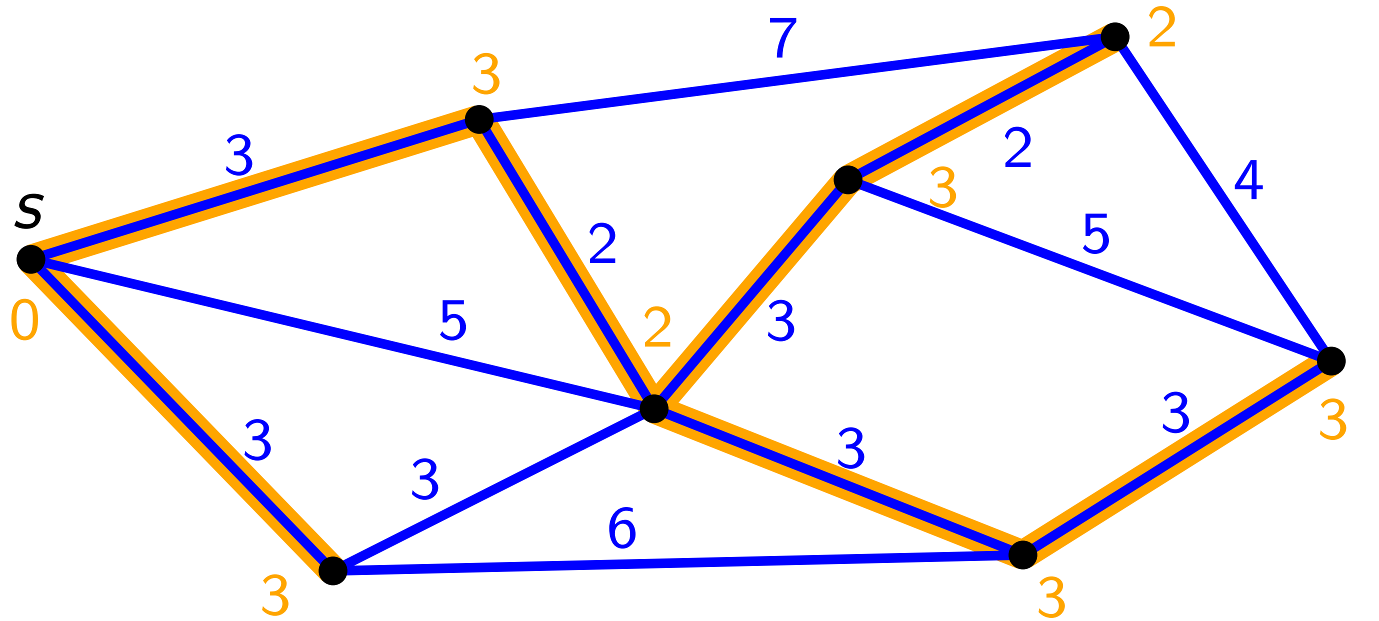
Übersicht

1. Graphdurchlaufstrategien
2. Kürzeste Wege
3. Minimale Spannbäume
   - 3.1 Jarník-Prim
     - Beispiel
     - Pseudocode
   - 3.2 Kruskal

Jarník-Prim – Beispiel

Eingabe: ungerichteter, zusammenhängender Graph $G(V, E)$ mit Kantengewichten und Knoten s

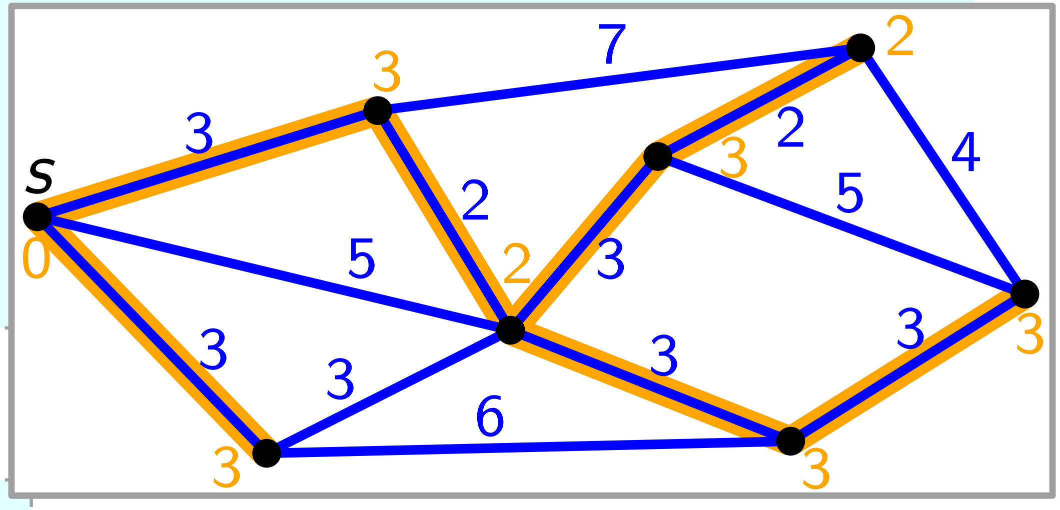
Ausgabe: gewichtsminimaler Baum T, der G aufspannt



Jarník-Prim – Pseudocode

JarníkPrimMST
~~Dijkstra~~(WeightedUndirectedGraph $G = (V, E; w)$, Vertex s)

- Initialize(G, s)
- $Q =$ **new** PriorityQueue(V, d)
- **while not** Q.Empty() **do**
  - $u = Q$.ExtractMin()
  - **foreach** $v \in \mathrm{Adj}[u]$ **do**
    - Relax$'$($u, v; w$)



$v \in Q$ **and** ...

Relax$'$($u, v; w$)

- **if** $v.d > \cancel{u.d +} w(u, v)$ **then**
  - $v.d = \cancel{u.d +} w(u, v)$
  - $v.\pi = u$
  - Q.UpdateKey($v, v.d$)

Laufzeit?

$\rightarrow$ siehe Dijkstra

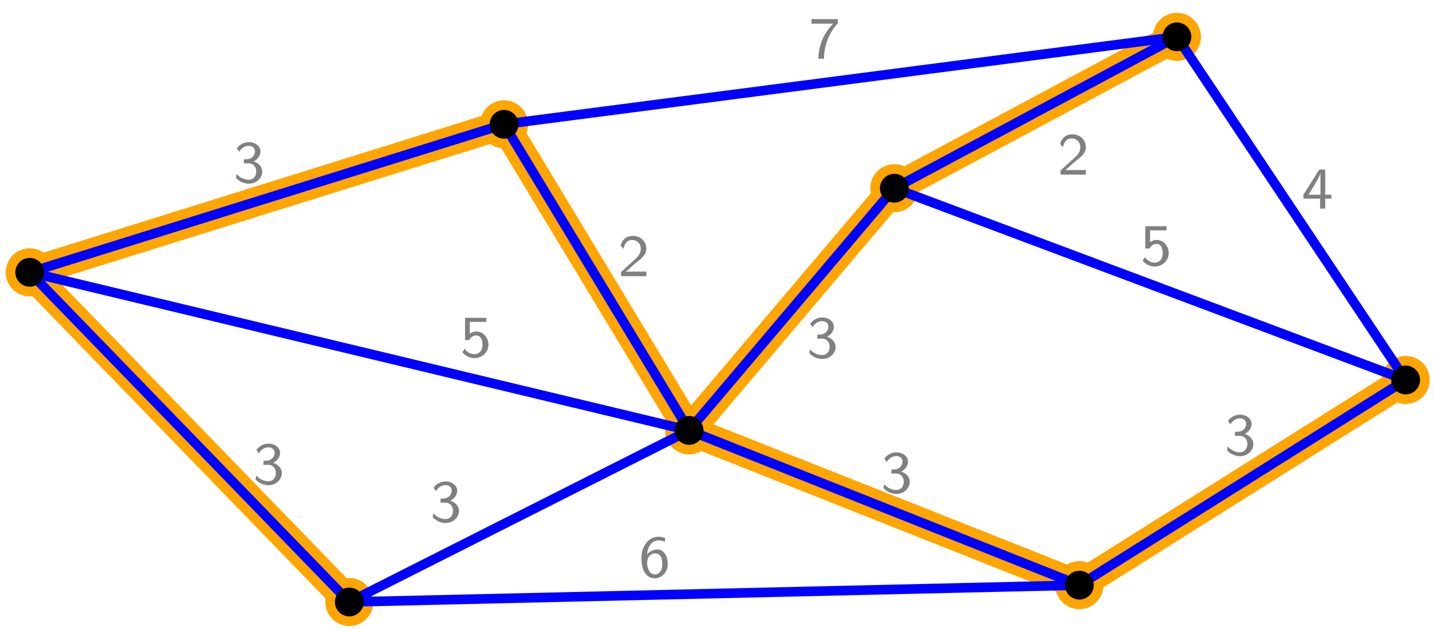
$\rightarrow O(E + V \log V)$

Übersicht

Kruskal – Beispiel

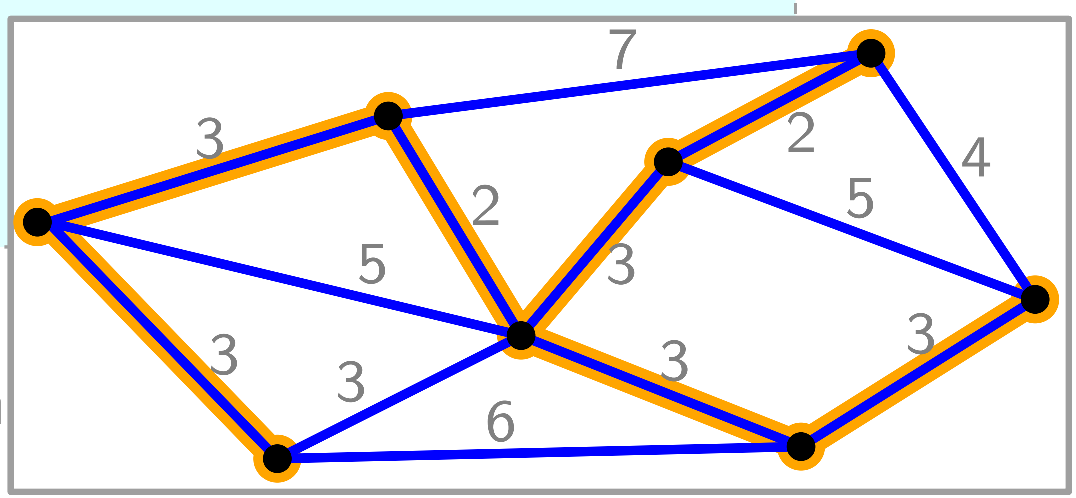
Eingabe: ungerichteter, zusammenhängender Graph $G(V, E)$ mit Kantengewichten

Ausgabe: gewichtsminimaler Baum T, der G aufspannt



Kruskal – Pseudocode

KruskalMST(WeightedUndirectedGraph $G = (V, E; w)$)
  $A = \emptyset$
  foreach $v \in V$ **do**
    MakeSet(v)
  Sortiere E nicht-absteigend nach Gewicht w
  foreach $uv \in E$ **do**
    if FindSet(u) $\neq$ FindSet(v) **then**
      $A = A \cup \{uv\}$
      Union(u, v)
  return A



Laufzeit?

$|V| \cdot \text{MakeSet} + (|V| - 1) \cdot \text{Union}$
$+ 2|E| \cdot \text{FindSet} + \text{Sort}(E)$
$= O(E \log V)$

Warum? Vergleiche ADS-Skript!

Viel Erfolg...

•

... beim Lösen des

1. Übungsblattes!