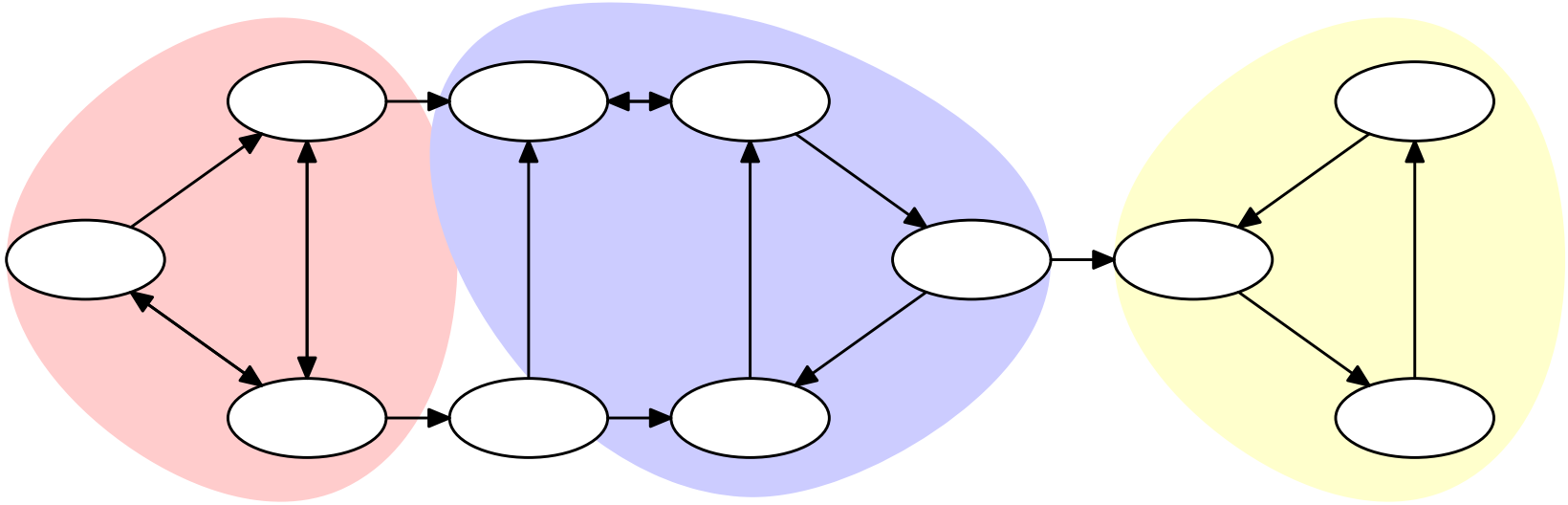
s-t Connectivity

Gegeben: Knoten s und t in gerichtetem Graph

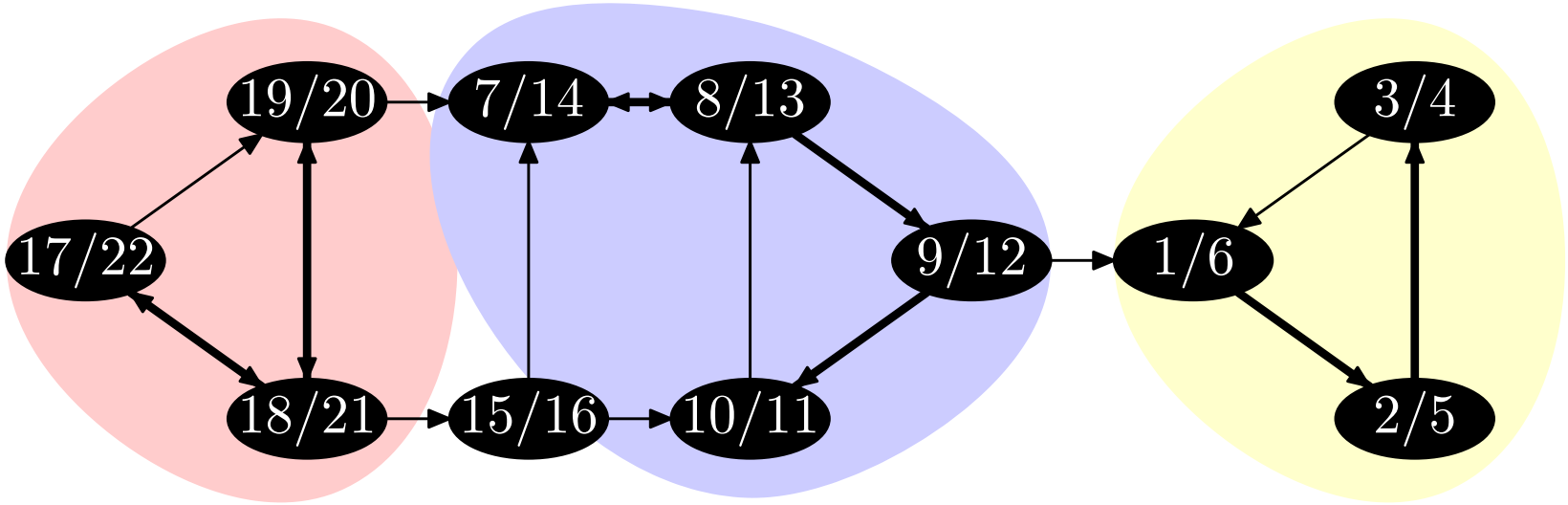
Frage: Ist t von s erreichbar (durch einen gerichteten Pfad)?



s-t Connectivity

Führe eine DFS aus, starte bei s.

Pfad von s nach $t \iff f(t) < f(s)$.

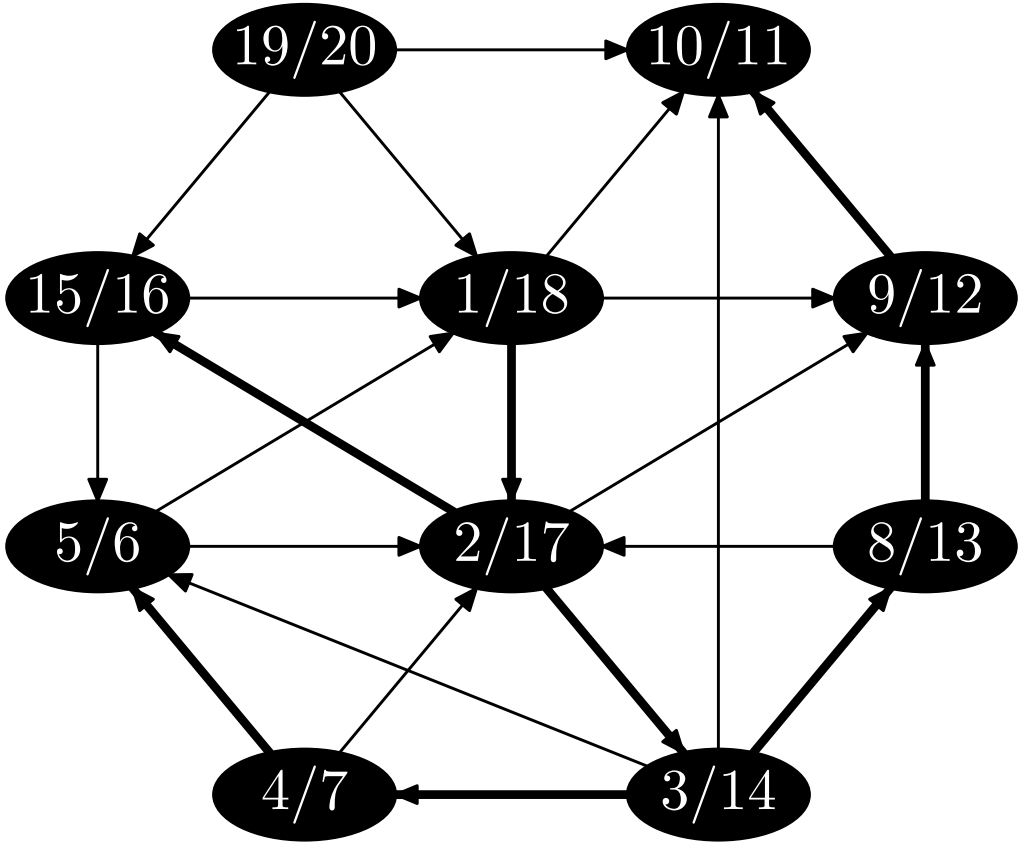


Beweis: Wenn s schwarz wird, sind alle von s erreichbaren Knoten schwarz.

s-t Connectivity

Theorem

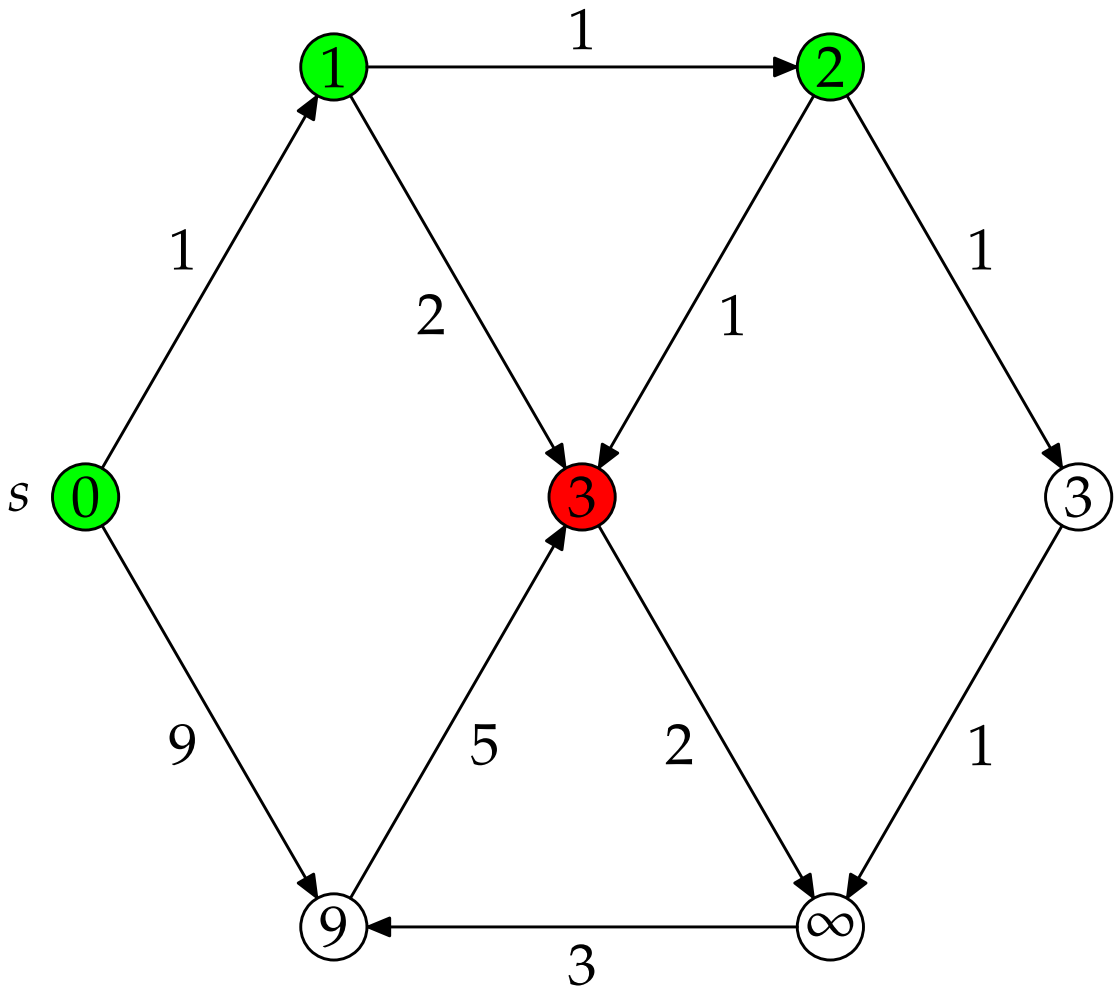
Gegeben sei ein gerichteter Graph $G = (V, E)$ und $s \in V$.
Wir können in linearer Zeit alle von s erreichbaren Knoten finden.



Single Source Shortest Paths

Gegeben ein gerichteter Graph $G = (V, E)$ mit nicht-negativen Kantengewichten $\mathit{length} : E \to \mathbf{Q}$ und ein Knoten $s \in V$, finde die kürzesten Wege von s zu allen Knoten.

- Wir lösen das Problem mit dynamischer Programmierung.
- Menge F von Knoten, deren Abstand bekannt ist.
- Anfangs ist $F = \{s\}$.
- F wird in jeder Iteration größer.
- Invariante: Kein Knoten $v \notin F$ hat kleineren Abstand zu s als jeder Knoten in F.



- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.

Korrektheit

Lemma

- *Jeder grüne Knoten enthält den Abstand von s.*
- *Jeder weiße Knoten enthält Abstand von s über grüne Knoten.*

Beweis.

Sei v einer weißer Knoten, dessen Beschriftung minimal ist.
Betrachte einen kürzesten Pfad von s nach v und den ersten weißen Knoten u auf diesem Pfad.
Es gilt $u = v$, da der Abstand von s zu u mindestens so groß ist wie der Abstand von s zu v.
Wenn ein Knoten grün wird, garantiert die Relaxation, daß die zweite Bedingung weiter gilt. □

Der Algorithmus von Dijkstra

Algorithmus

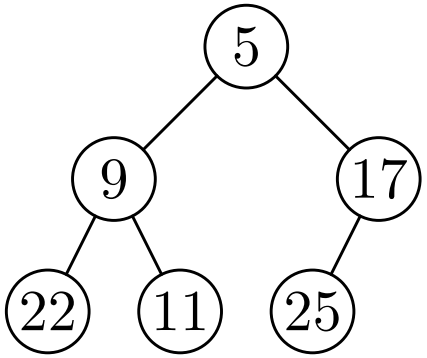
procedure *Dijkstra*(s) :
$Q := V - \{s\}$;
for $v \in Q$ **do** $d[v] := \infty$ **od**;
$d[s] := 0$;
while $Q \neq \emptyset$ **do**
  choose $v \in Q$ *with minimal* $d[v]$;
  $Q := Q - \{v\}$;
  forall u *adjacent* **to** v **do**
    $d[u] := min\{d[u], d[v] + length(v, u)\}$
  od
od

Wie implementieren wir Q?

Priority Queues (Prioritätswarteschlangen)

Operationen einer Prioritätswarteschlange Q:

1. Einfügen von x mit Gewicht w (insert)
2. Finden und Entfernen eines Elements mit minimalem Gewicht (extract-min)
3. Das Gewicht eines Elements x auf w verringern (decrease-weight)



Heap: alle Operationen in $O(\log n)$ Schritten
(n ist die aktuelle Anzahl von Elementen im Heap)

Algorithmus von Dijkstra – Laufzeit

Theorem

Der Algorithmus von Dijkstra berechnet die Abstände von s zu allen anderen Knoten in $O((|V| + |E|) \log |V|)$ Schritten.

Beweis.

Es werden $|V|$ Einfügeoperationen, $|V|$ extract-mins und $|E|$ decrease-keys ausgeführt.
Verwenden wir einen Heap für die Prioritätswarteschlange, ergibt sich die verlangte Laufzeit. □

Ein Fibonacci-Heap benötigt für ein Einfügen und extract-min $O(\log n)$ und für decrease-key nur $O(1)$ amortisierte Zeit.

Dijkstra: $O(|V| \log |V| + |E|)$

Java

```
public void dijkstra(Node s, Map<Node, Node> pred) {
  SimpleIterator<Node> it;
  for(it = nodeiterator(); it.more(); it.step()) it.key().weight = 1e10;
  s.weight = 0.0;
  Heap<Node> H = new Heap<Node>();
  for(it = nodeiterator(); it.more(); it.step()) H.insert(it.key());
  while(!H.isempty()) {
     Node v = H.extract_min();
     Iterator<Node, Edge> inc;
     for(inc = neighbors.find(v).iterator(); inc.more(); inc.step())
        if(inc.key().weight > v.weight + inc.data().weight) {
           pred.insert(inc.key(), v);
           inc.key().weight = v.weight + inc.data().weight;
           H.decrease_key(inc.key());
        }
  }
}
```

Spezialfall DAG

Können wir kürzeste Pfade in DAGs schneller finden?

Theorem

Kürzeste Pfade von einem Knoten s in einem DAG können in linearer Zeit gefunden werden.

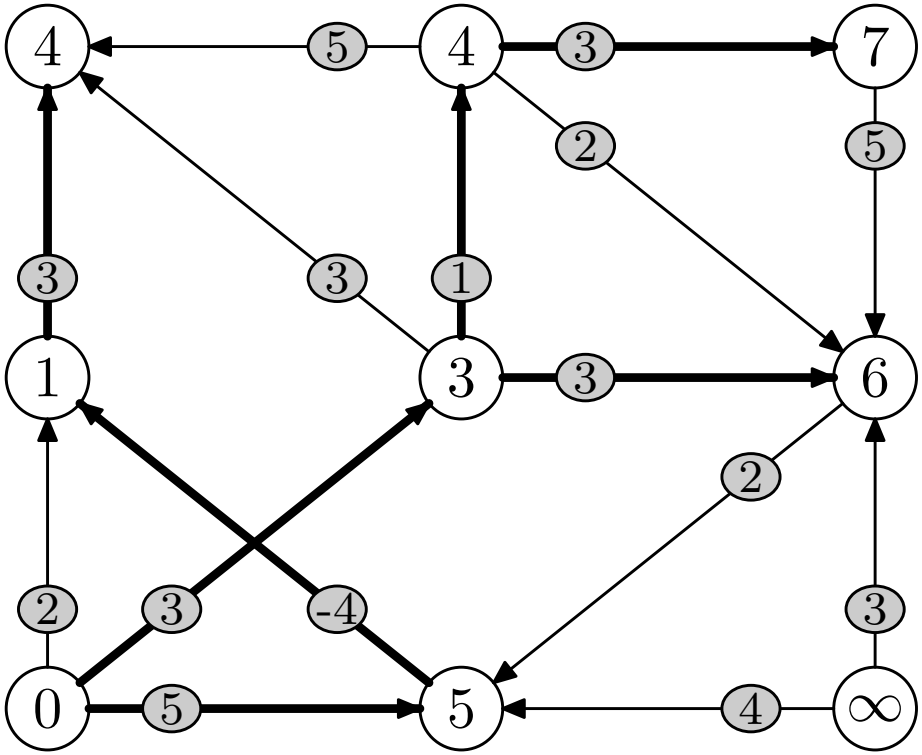
Beweis.

Relaxiere Knoten in topologischer Reihenfolge.

Laufzeit: $O(|V| + |E|)$.
Korrektheit: Jeder Knoten, der relaxiert, kennt zu diesem Zeitpunkt seinen echten Abstand. □

Frage: Sind negative Gewichte hier erlaubt?

Kürzeste Wege mit negativen Kantengewichten



Dijkstra funktioniert nicht mit negativen Kantengewichten.

Der Algorithmus von Bellman und Ford

Idee:

- Relaxiere alle Kanten
- Wiederhole dies, bis keine Änderung

Warum korrekt?

Induktion über die Länge eines kürzesten Pfads.
(Also genügen n Wiederholungen)

Was passiert bei Kreisen mit negativem Gewicht?

$\rightarrow$ Keine Terminierung.

Der Algorithmus von Bellman und Ford

Algorithmus

function $Bellman - Ford(s)$ **boolean** :
for $v \in V$ **do** $d[v] := \infty$ **od**;
$d[s] := 0$;
for $i = 1$ **to** $|V| - 1$ **do**
  forall $(u, v) \in E$ **do**
    $d[u] := min\{d[u], d[v] + length(v, u)\}$
  od
od;
forall $(u, v) \in E$ **do**
  if $d[u] > d[v] + length(v, u)$ **then return** *false* **fi**
od;
return *true*

Der Algorithmus von Bellman und Ford

Theorem

Gegeben sei ein gerichteter Graph $G = (V, E)$ mit Kantengewichten $E \to \mathbf{R}$ und ein Knoten $s \in V$.
Wir können in $O(|V| \cdot |E|)$ feststellen, ob ein Kreis mit negativem Gewicht existiert, und falls nicht, die kürzesten Wege von s zu allen Knoten berechnen.

Beweis.

Wir haben die Korrektheit bereits nachgewiesen.
Zur Laufzeit: Jeder Knoten wird $|V|$ mal relaxiert, also wird auch jede Kante $|V|$ mal relaxiert (in konstanter Zeit). □