Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Kürzeste Wege

Zentrale Frage:

Wie komme ich am schnellsten von A nach B in einem Graphen, in dem Kanten Kosten zugeordnet werden?

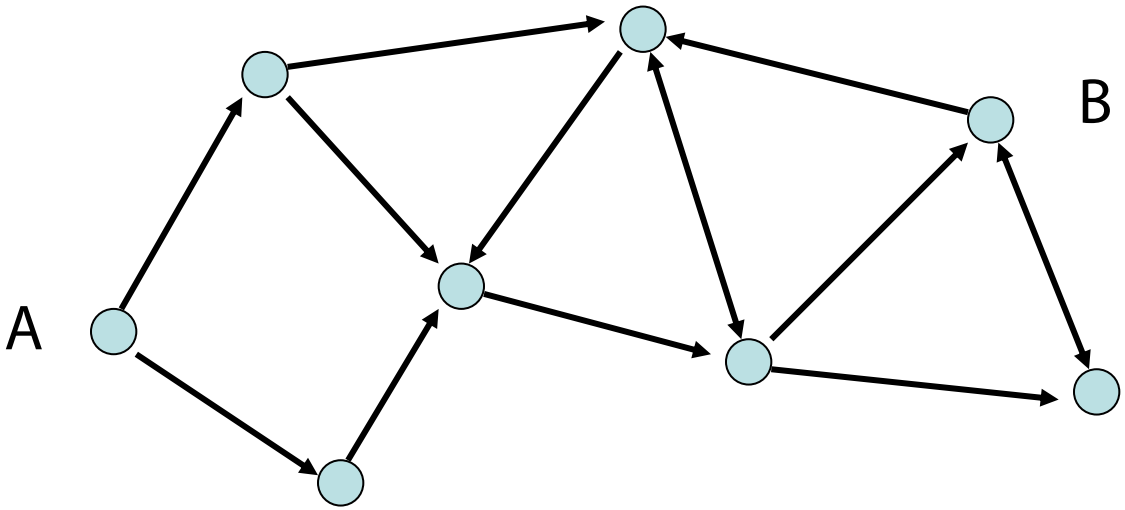
Fälle:

- Kantenkosten 1
- DAG, beliebige Kantenkosten
- Beliebiger Graph, positive Kantenkosten
- Beliebiger Graph, beliebige Kosten

Kürzeste Wege

Zentrale Frage:

Wie komme ich am schnellsten von A nach B?



Kürzeste Wege

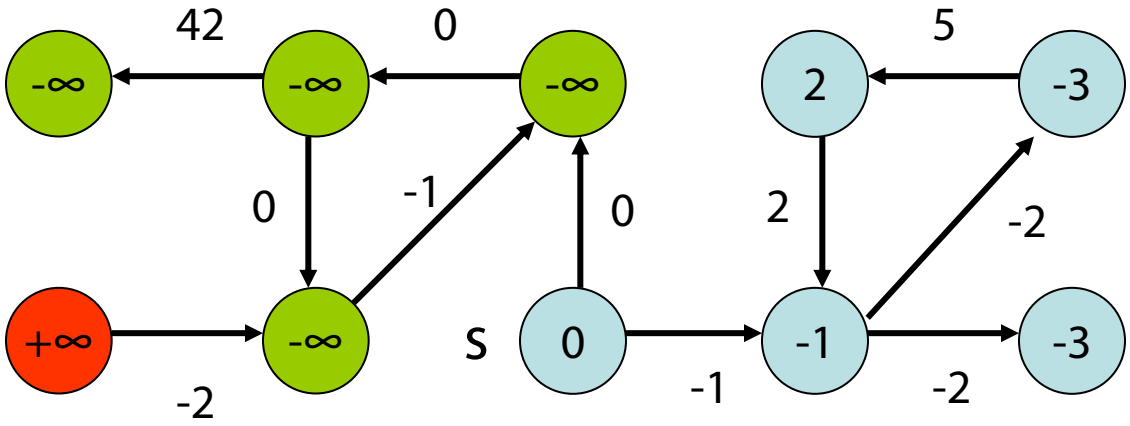
Kürzeste-Wege-Problem:

- gerichteter Graph $G = (V, E)$
- Kantenkosten $c : E \longrightarrow \mathbb{R}$

- SSSP (single source shortest path):
  Kürzeste Wege von einer Quelle zu allen anderen Knoten
- APSP (all pairs shortest path):
  Kürzeste Wege zwischen allen Paaren

H. C. Joksch. *The shortest route problem with constraints*.
J. Math. Analysis and Applications. 14, Seite 191–197, **1966**

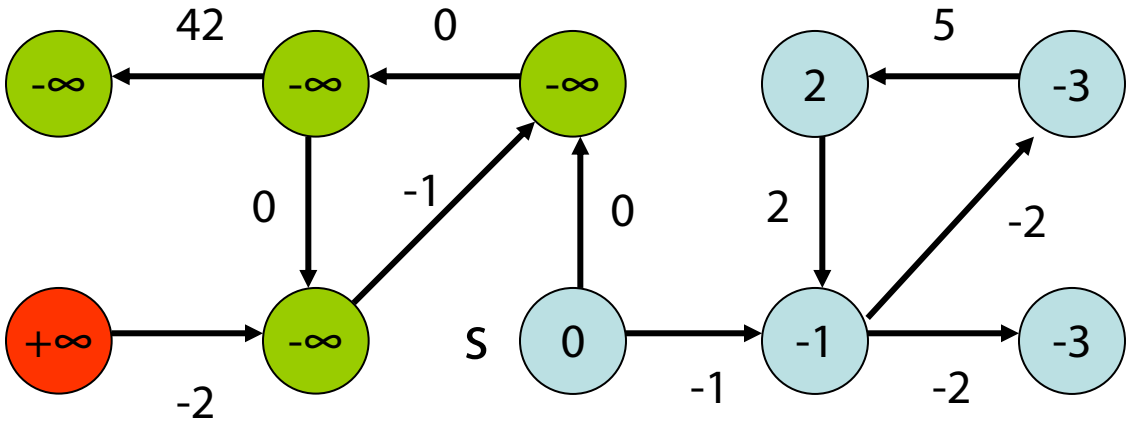
Kürzeste Wege



$\mu(s,v)$: Distanz zwischen s und v

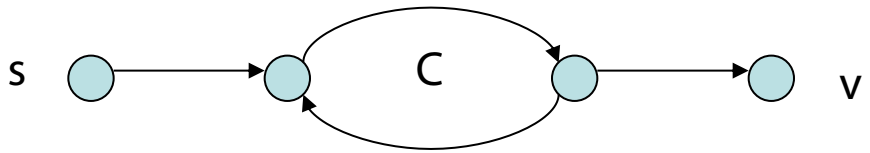
$$\mu(s,v) = \begin{cases} \infty & \text{kein Weg von s nach v} \\ -\infty & \text{Weg bel. kleiner Kosten von s nach v} \\ \min\{ c(p) \mid p \text{ ist Weg von s nach v}\} & \end{cases}$$

Kürzeste Wege



Wann sind die Kosten -∞?

Wenn es einen negativen Kreis gibt:

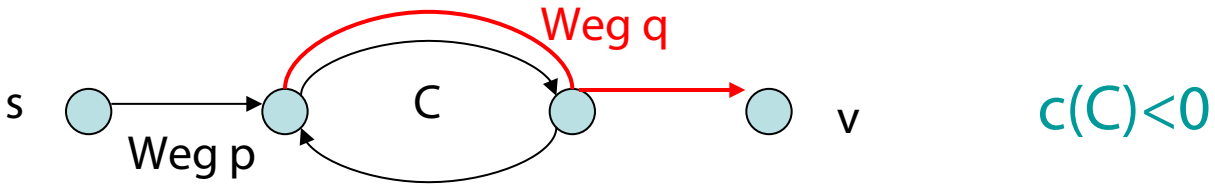


c(C)<0

Kürzeste Wege

Negativer Kreis hinreichend und notwendig für Wegekosten $-\infty$.

Negativer Kreis hinreichend:

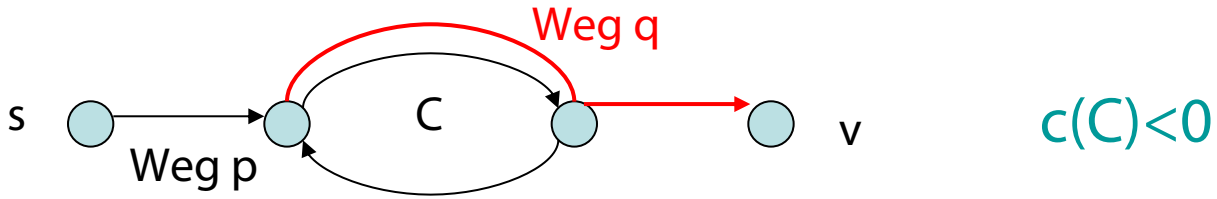


Kosten für i-fachen Durchlauf von C:
$c(p) + i \cdot c(C) + c(q)$
Für $i \longrightarrow \infty$ geht Ausdruck gegen $-\infty$.

Kürzeste Wege

Negativer Kreis notwendig:

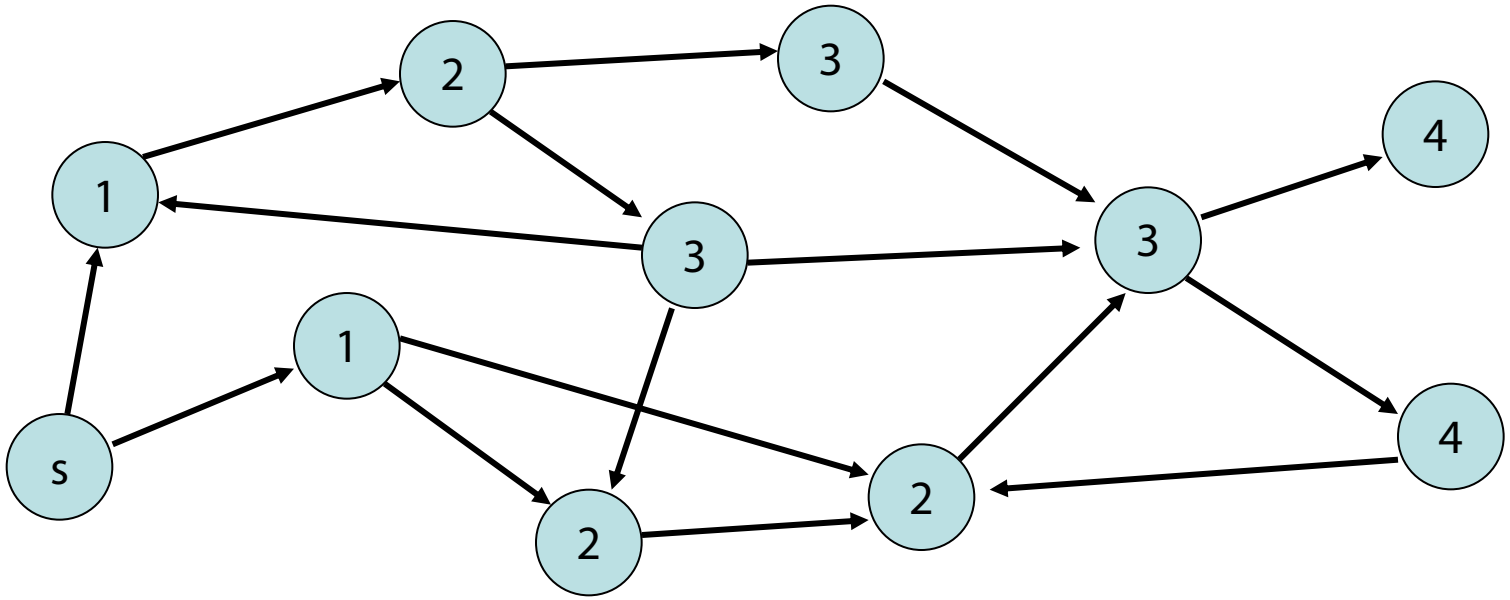
- Kosten $v=-\infty$, also Kreis C vorhanden
- l: minimale Kosten eines einfachen Weges von s nach v
- Es gibt nicht einfachen Weg r von s nach v mit Kosten $c(r)<l$
- r nicht einfach: Zerlegung von r in pCq, wobei C ein Kreis ist und pq ein einfacher Weg
- Da $c(r) < l \leq c(pq)$ ist, gilt $c(C)<0$



Kürzeste Wege in Graphen

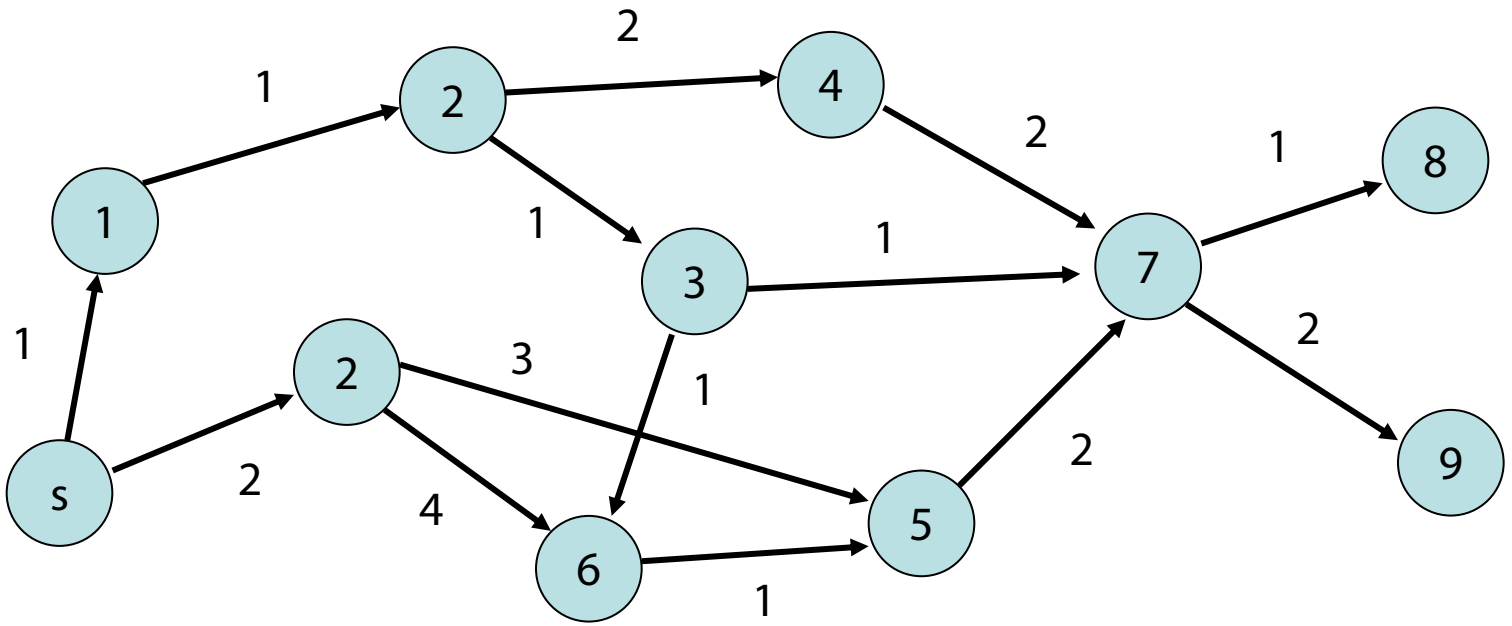
Graph mit Kantenkosten 1:

Führe Breitensuche durch.



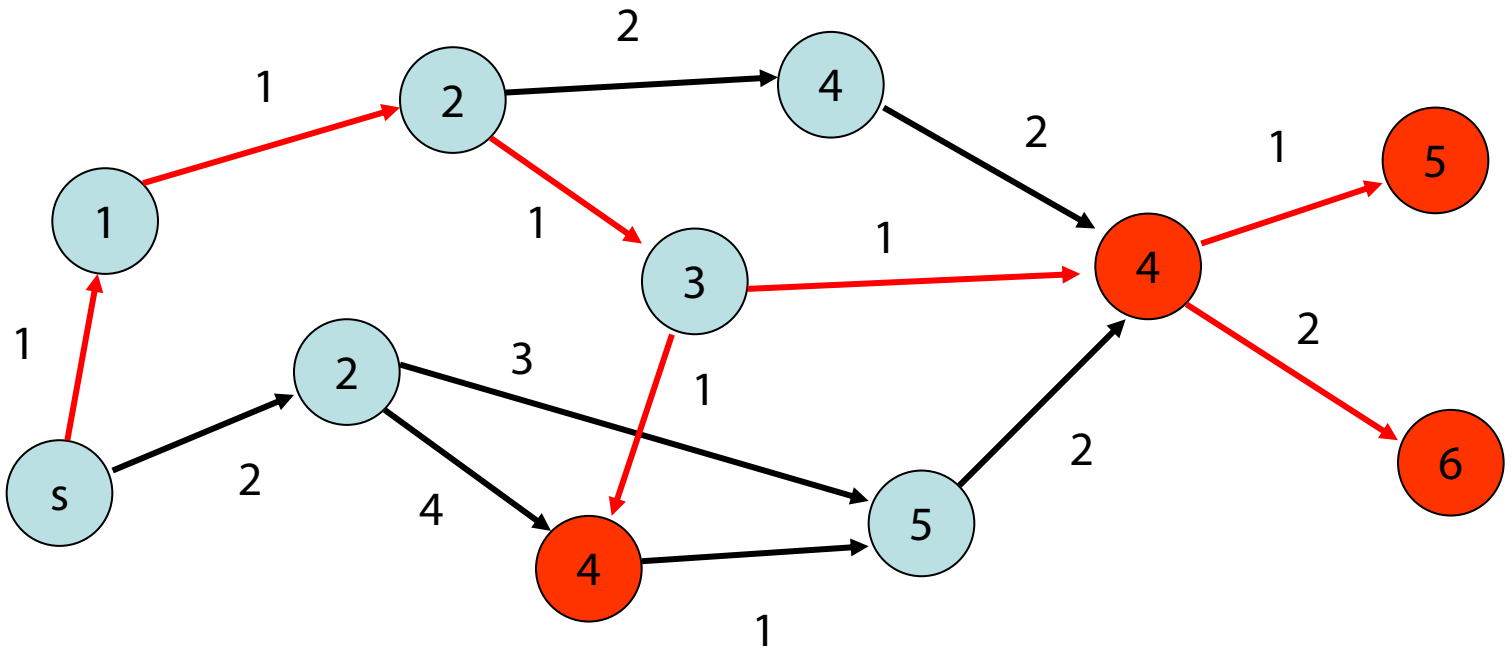
Kürzeste Wege in DAGs

- Reine Breitensuche funktioniert nicht, wenn Kantenkosten nicht gleich 1.



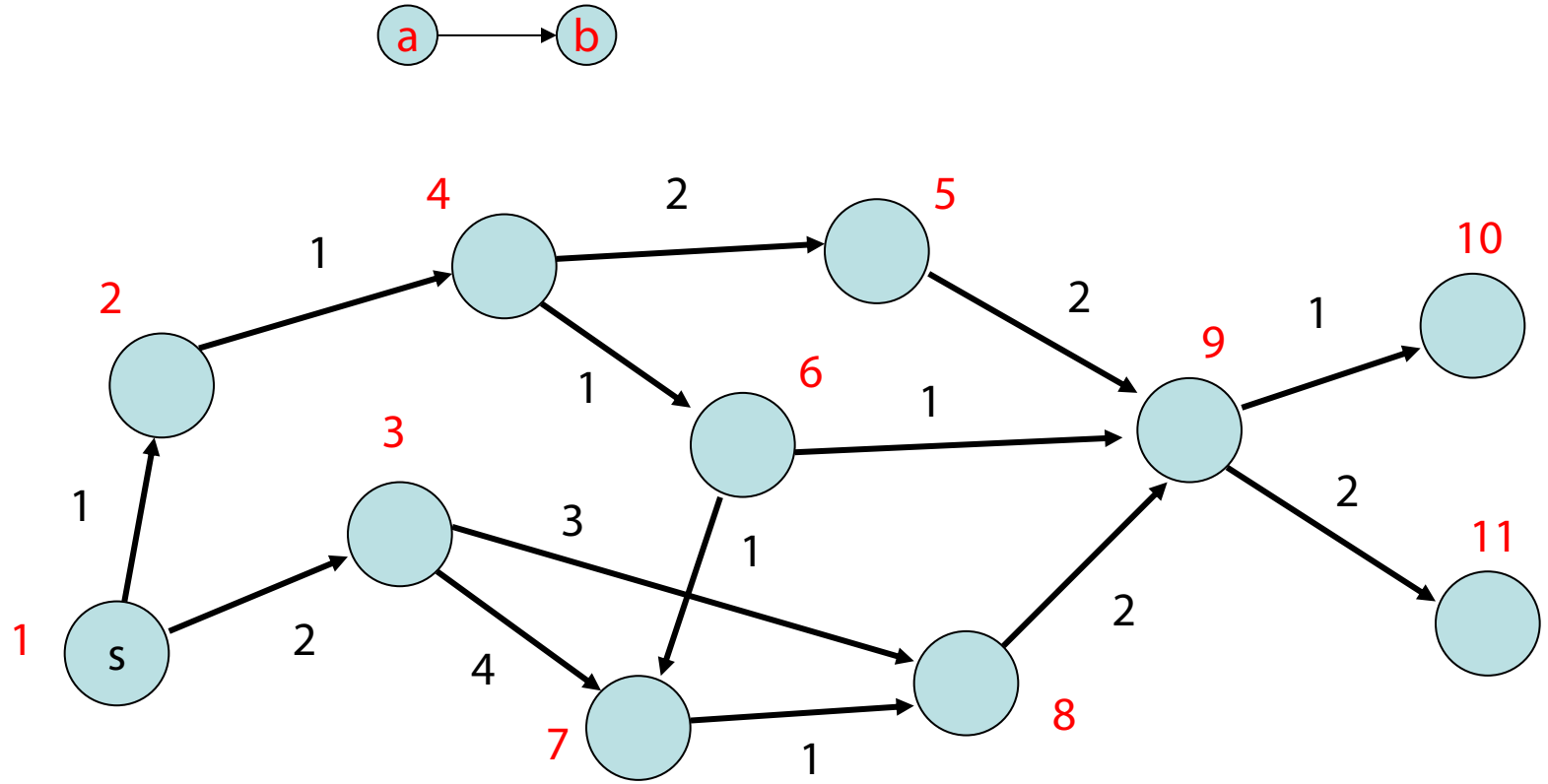
Kürzeste Wege in DAGs

Korrekte Distanzen:



Kürzeste Wege in DAGs

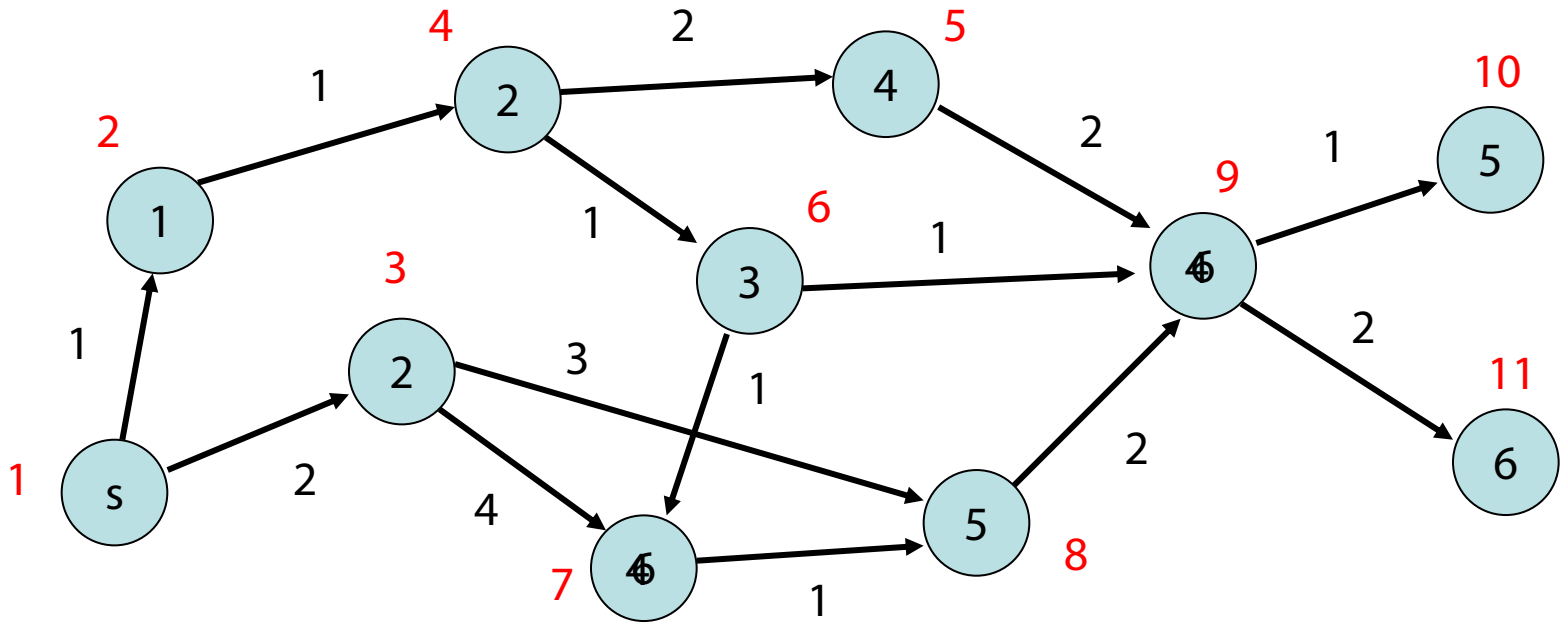
Strategie: nutze aus, dass Knoten in DAGs topologisch sortiert werden können (alle Kanten erfüllen a<b)



Kürzeste Wege in DAGs

Strategie:

Betrachte dann Knoten in der Reihenfolge ihrer topologischen Sortierung und aktualisiere Distanzen zu s



Kürzeste Wege in DAGs

Strategie:

1. Topologische Sortierung der Knoten
2. Aktualisierung der Distanzen gemäß der topologischen Sortierung

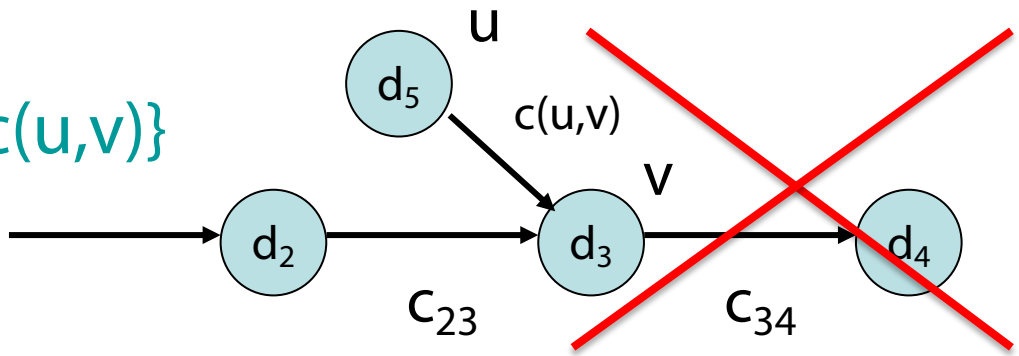
Warum funktioniert das??

Kürzeste Wege in Graphen

Allgemeine Strategie:

- Am Anfang, setze d(s):=0 und d(v):=∞ für alle Knoten v ∈ V \ {s}
- Für jeden besuchten Knoten u, aktualisiere die Distanzen der Knoten v mit (u,v)∈ E,

  d(v) := min{d(v), d(u)+c(u,v)}



- Besuche Knoten in einer Reihenfolge, die sicherstellt, dass mindestens ein kürzester Weg von s zu jedem v gefunden ist, bevor v expandiert wird

Kürzeste Wege in DAGs

Es gilt:

Expansion in topologischer Reihenfolge führt zu richtigen Distanzen

Zurück zur Strategie:

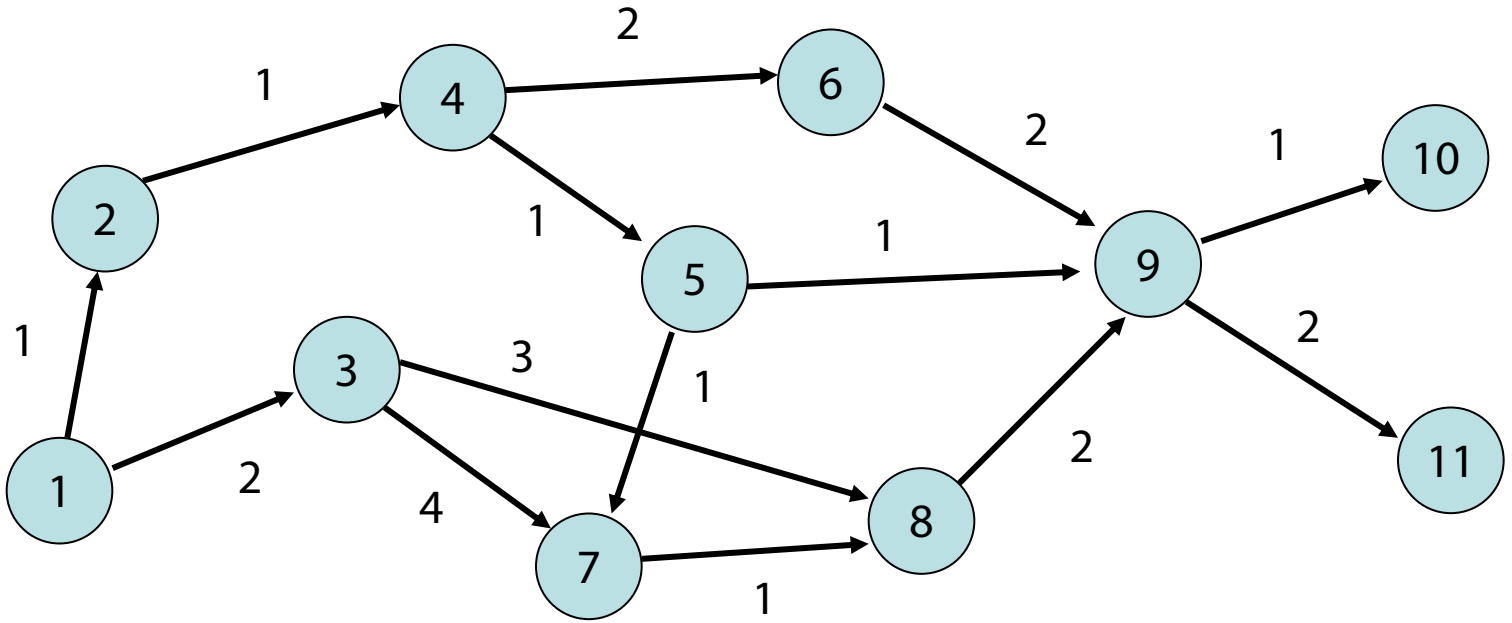
1. Topologische Sortierung der Knoten
2. Aktualisierung der Distanzen gemäß der topologischen Sortierung

Wie führe ich eine topologische Sortierung durch?

Kürzeste Wege in DAGs

Beispiel:

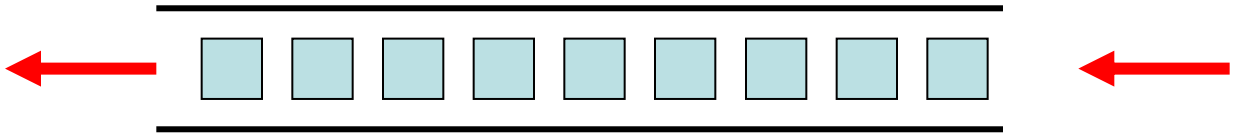
- : Knoten momentan in Queue q
- Nummerierung nach Einfügereihenfolge



Kürzeste Wege in DAGs

Topologische Sortierung:

- Verwende eine FIFO Queue q und Zähler n := 1

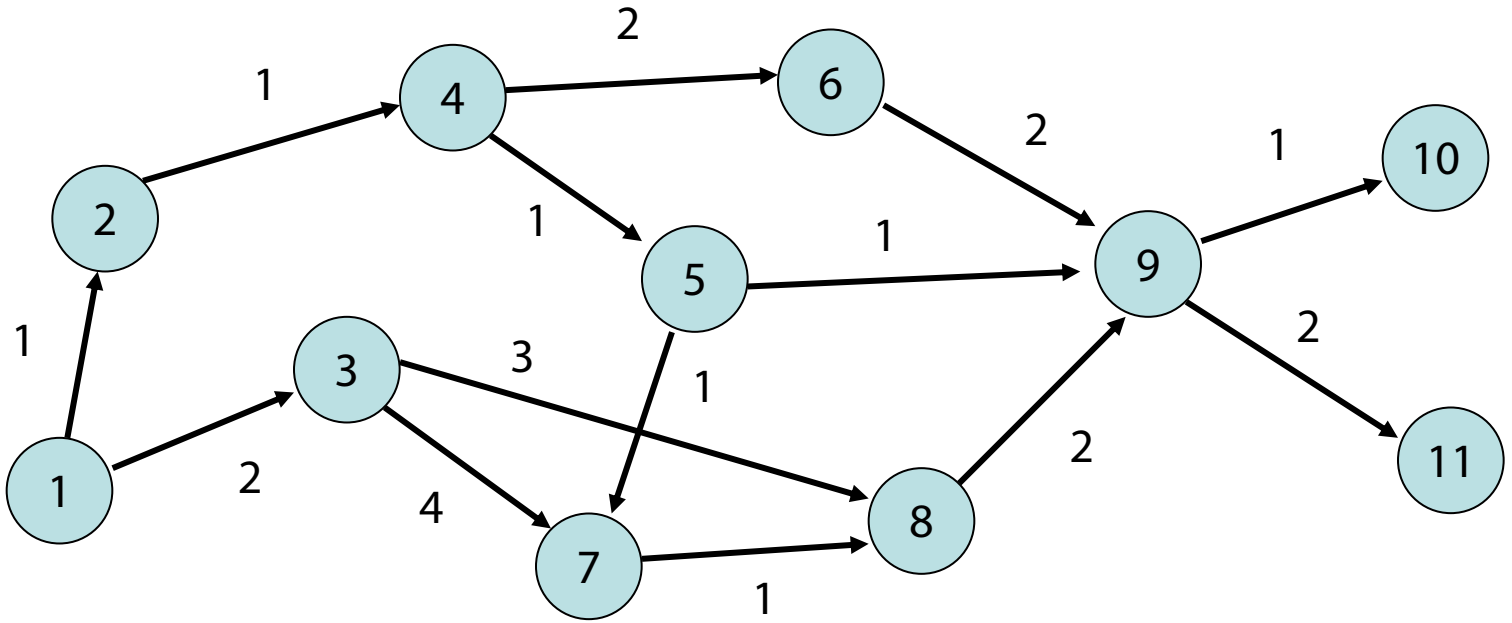


- Bei Einfügen von v in q: num(v) := n; n := n+1
- Anfangs enthält q alle Knoten, die keine eingehende Kante haben (Quellen).
- Solange q nicht leer
  - Entnehme u aus q und markiere alle (u, v) ∈ E. Falls alle Kanten nach v markiert, füge v in q ein.
- Erfolg, falls alle Knoten nummeriert

Kürzeste Wege in DAGs

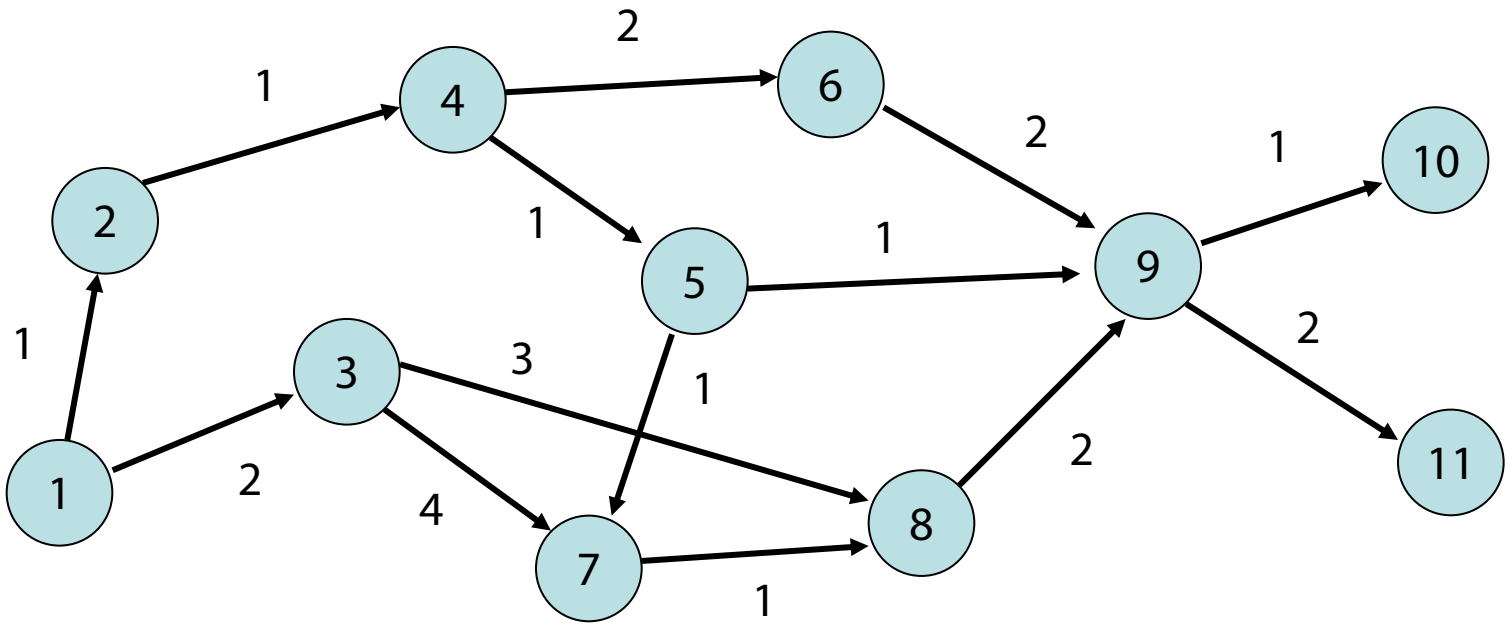
Korrektheit der topologischen Nummerierung:

Knoten wird erst dann nummeriert, wenn alle Vorgänger nummeriert sind.



Kürzeste Wege in DAGs

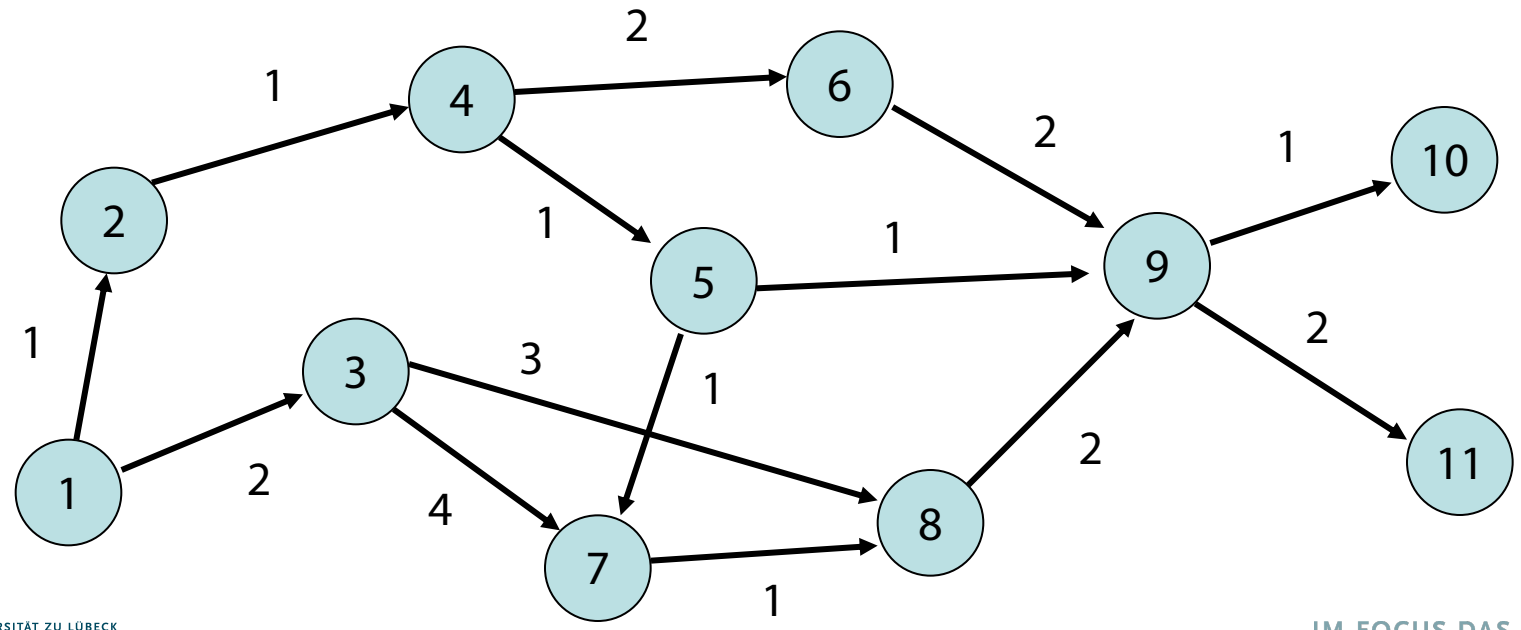
Laufzeit: Zur Bestimmung aller Knoten ohne eingehende Kante muss Graph einmal durchlaufen werden. Danach wird jeder Knoten und jede Kante genau einmal betrachtet, also Zeit O(n+m).



Kürzeste Wege in DAGs

Bemerkung: topologische Sortierung kann nicht alle Knoten nummerieren genau dann, wenn Graph gerichteten Kreis enthält (kein DAG ist)

Test auf DAG-Eigenschaft



Kürzeste Wege in DAGs

DAG-Strategie:

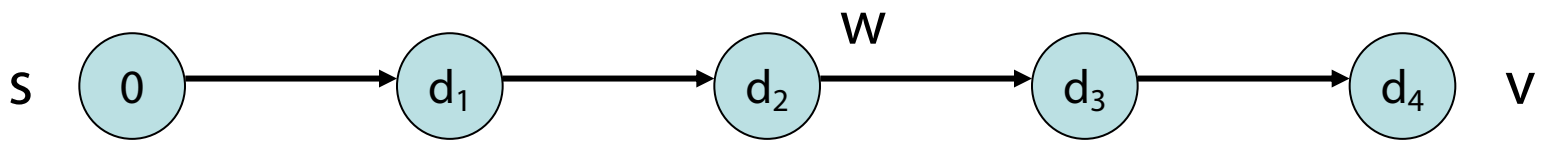
1. Topologische Sortierung der Knoten
   Laufzeit O(n+m)
2. Aktualisierung der Distanzen gemäß der topologischen Sortierung
   Laufzeit O(n+m)

Insgesamt Laufzeit O(n+m).

Dijkstras Algorithmus

Nächster Schritt: Kürzeste Wege für beliebige Graphen mit positiven Kanten.

Problem: besuche Knoten eines kürzesten Weges in richtiger Reihenfolge



Lösung: besuche Knoten in der Reihenfolge der kürzesten Distanz zur Quelle s

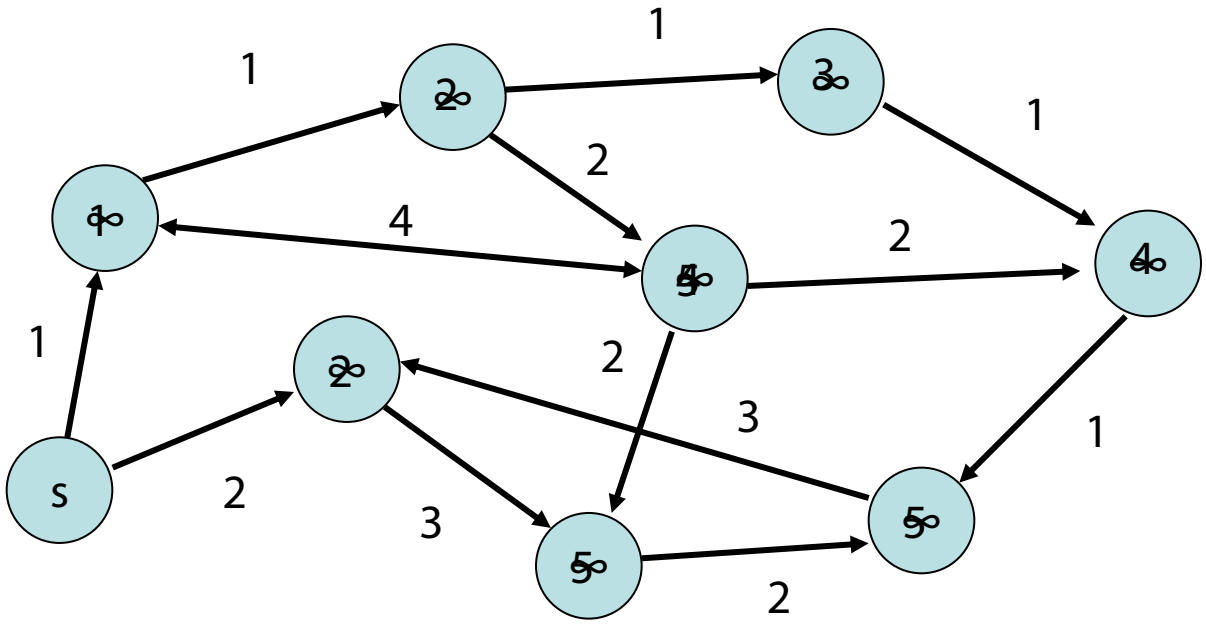
Edsger W. Dijkstra: A note on two problems in connexion with graphs. In: Numerische Mathematik. 1, S. 269–271, **1959**

Dijkstras Algorithmus

- Am Anfang, setze $d(s):=0$ und $d(v):=\infty$ für alle Knoten. Füge s in Prioritätswarteschlange q ein, wobei die Prioritäten in q gemäß der aktuellen Distanzen $d(v)$ von s zu v definiert sind.
- Wiederhole, bis q leer:
  - Entferne mittels deleteMin(q) aus q den Knoten u mit niedrigstem $d(u)$
  - Für alle $(u, v) \in E$, setze $d(v) := \min\{d(v), d(u)+c(u,v)\}$.
  - Falls v noch nicht in q war,
    - dann füge v in q ein,
    - sonst verringere Kosten von v mit decreaseKey(v, q, Δ), sofern $\Delta < 0$

Dijkstras Algorithmus

Beispiel: (: aktuell, : fertig)



Dijkstras Algorithmus

```
procedure Dijkstra(s: NodeId)
    d=<∞,…,∞>: NodeArray of ℝ ∪ {-∞,∞}
    d[s]:=0
    q=<s>: PQ mit Schlüssel d so dass d(x) = d[x]
    while not mtQueue?(q) do
       u := deleteMin(q)  // u: min. Distanz zu s in q
       for e=(u,v) ∈ E do
          dv := d[v];
          if dv = ∞ then insert(v, q) // v schon in q?
          else if dv > d[u]+c(e) then // aktualisiere d[v]
                   d[v]:=d[u]+c(e)
                   decreaseKey(v, q, dv-d[v])
```

Dijkstras Algorithmus

Laufzeit:

$T_{Dijkstra} \in O(n(T_{DeleteMin}(n)+T_{Insert}(n)) + m \cdot T_{decreaseKey}(n))$

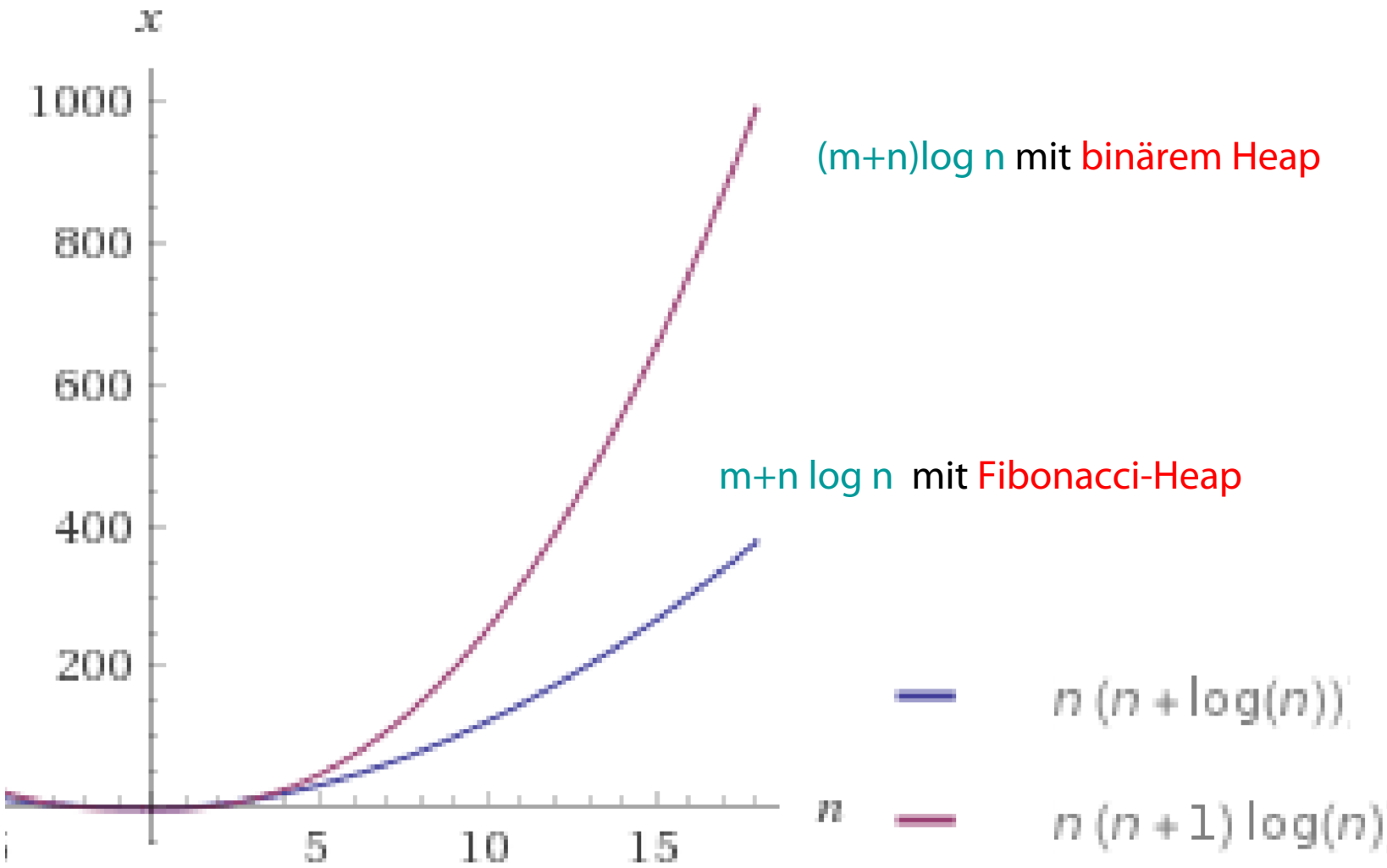
Binärer Heap: alle Operationen $O(\log n)$,

also $T_{Dijkstra} \in O((m+n)\log n)$

Fibonacci Heap:

- $T_{DeleteMin}(n) = T_{Insert}(n) \in O(\log n)$
- $T_{decreaseKey}(n) \in O(1)$
- Damit $T_{Dijkstra} \in O(n \log n + m)$

Dijkstra-Algorithmus: Vergleich mit $m = n^2$



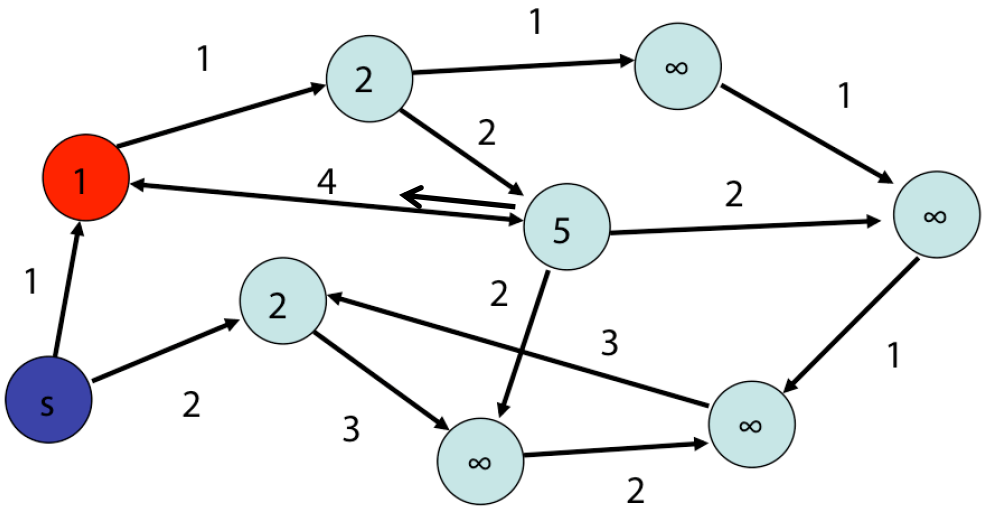
(m+n)log n mit binärem Heap

m+n log n mit Fibonacci-Heap

Kürzeste Wege

- Nachteil der bisherigen Verfahren:
  - Nur Länge des kürzesten Weges bestimmt
  - Zur Wegebestimmung muss Rückzeiger verwaltet werden

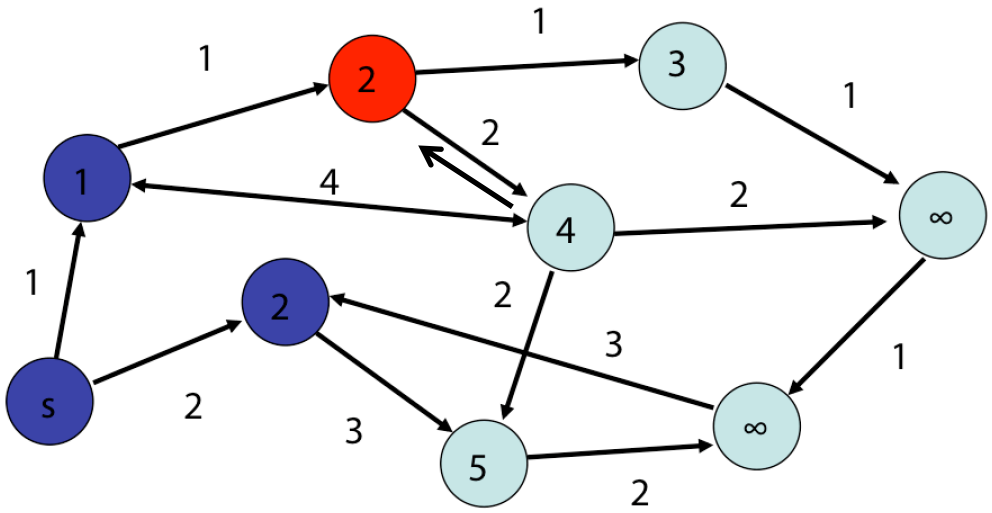
Beispiel: (: aktuell, : fertig)



Kürzeste Wege

- Nachteil der bisherigen Verfahren:
  - Nur Länge des kürzesten Weges bestimmt
  - Zur Wegebestimmung muss Rückzeiger verwaltet werden

Beispiel: (: aktuell, : fertig)



Zusammenfassung

- Single-Source Shortest Paths
- Negative Kreise
- Kürzeste Weg in gerichteten Graphen
  - Topologische Sortierung
  - Dijkstra's Algorithmus