02.12.09

Wir haben $d(s,v) = d_{(R,T)}(s,v)$,

aber $d(s,w) < d_{(R,T)}(s,w)$,

also gehört $e$ nicht zu $T$.

Außerdem ist

$$\ell(w) = d_{(R,T)}(s,w) > d(s,w) = d(s,v) + 1$$
$$= d_{(R,T)}(s,v) + 1 = \ell(v) + 1.$$

Wäre $w \in R$, hätten wir wegen

$$\ell(w) > \ell(v) + 1$$

einen Widerspruch zur Eigenschaft (*)

Also muss $w \notin R$ gelten; dann verbindet aber die Kante $e = \{v,w\}$ zum Zeitpunkt der Entfernung von $v$ aus $Q$ $v$ mit einem Knoten $w \notin R$, im Widerspruch zur Abfrage in (3).

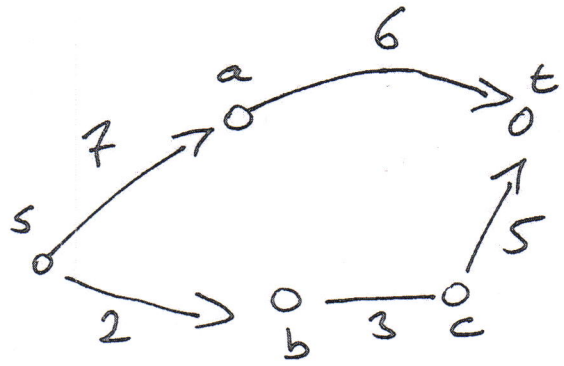
□

# 3.8 Ausblick: Algorithmische Probleme auf Graphen

**Problem 1:** Kürzeste Weg mit Kantenlängen

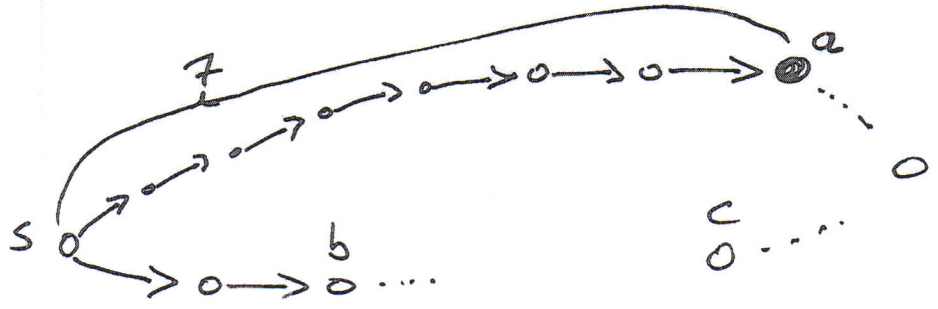
**Gegeben:** Graph $G=(V,E)$, $c: E \to \mathbb{N}$ Kantenlänge, zwei Knoten $s,t \in V$.

**Gesucht:** Ein kürzester Weg von $s$ nach $t$.




Würde man die Kantenlängen ignorieren und BFS anwenden wäre das Ergebnis 13 (und damit nicht optimal)

Ersetzt man eine Kante der Länge $c(e)$ durch $c(e)$ Kanten der Länge 1, könnte man auf diesen Graphen BFS anwenden.

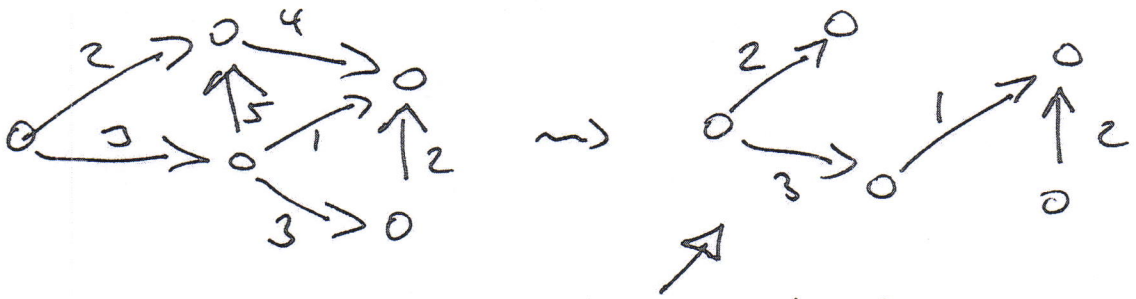



Allerdings wird in diesem Graphen die Laufzeit von BFS ziemlich schlecht.

# Problem 2: Kostengünstigste Netzwerke

**Gegeben:** Graph $G=(V,E)$. $c: E \to \mathbb{N}$ Kantengewichte

**Gesucht:** Ein zusammenhängender Teilgraph $T$ mit $\sum_{e \in T} c(e)$ minimal.




Hat man pos. Kantengewichte, gibt es in der Lösung nie einen Kreis (man könnte eine Kante weglassen und hätte eine bessere Lösg)

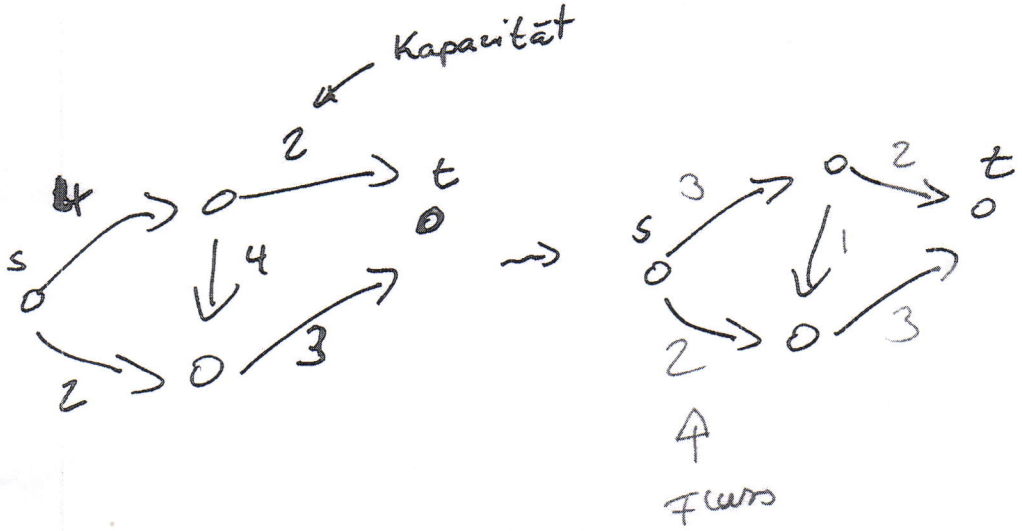
Deshalb spricht man auch von:

Minimalen aufspannenden Bäumen.

# Problem 3: Flussprobleme

**Gegeben:** Graph $G=(V,E)$. Kapazitäten der Kanten: $c: E \to \mathbb{N}$. Zwei Knoten $s,t$.

**Gesucht:** Maximaler Fluss von $s$ nach $t$ (der die Kapazitäten auf den Kanten nicht überschreitet)




Knoten $v \in V \setminus \{s,t\}$ leiten das Gut nur weiter.

Beispiele
- Verkehr (evtl. mehrere „Quellen")
- Datenpakete

# 4 Einfache Datenstrukturen

## 4.1 Grundoperationen

Aufgabenstellung:
- Verwalten einer Menge $S$ von Objekten
- Ausführen von verschiedenen Operationen (s.u.!)

Im Folgenden:

| | | |
|---|---|---|
| $S$ | Menge | |
| $k$ | Wert eines Elements | (Name, Größe, d.h. Speicherinhalt) |
| $x$ | Zeiger auf Element | (d.h. Speicheradresse) |
| NIL | spezieller, „leerer" Zeiger | |

Dann:

SEARCH $(S, k)$: „Suche in $S$ nach $k$"

Durchsuche die Menge $S$ nach einem Element von Wert $k$.

Ausgabe: Zeiger $x$, falls $x$ existent
NIL, falls kein Element wert $k$ hat

INSERT $(S, x)$: „Füge $x$ in $S$ ein"

Erweitere $S$ um das Element, das unter der Adresse $x$ steht.

DELETE $(S, x)$: „Entferne $x$ aus $S$"

Lösche das unter der Adresse $x$ stehende Element aus der Menge $S$.

MINIMUM $(S)$: „Suche das Minimum von $S$"

Finde in $S$ ein Element von kleinstem Wert.
(Annahme: Die Werte lassen sich vollständig ordnen)

Ausgabe: Zeiger $x$ auf so ein Element

MAXIMUM $(S)$: „Suche das Maximum von $S$"

Finde in $S$ ein Element von größtem Wert.
(Annahme: Werte lassen sich ordnen)

Ausgabe: Zeiger $x$ auf Element.

SUCCESSOR $(S, x)$: „Finde das nächstgrößere Element"

Für ein in $x$ stehendes Element in $S$,
bestimme ein ~~nächstgrößeres~~ Element von
nächstgrößerem Wert in $S$.
(Annahme: Totalordnung)

Ausgabe: Zeiger $y$ auf Element
NIL, wenn $x$ Maximum von $S$ ergibt