Algorithmen und Datenstrukturen

SoSe 2008 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Algorithmen und Datenstrukturen

Gesamtübersicht

- Organisatorisches / Einführung
- Grundlagen: RAM, $\mathcal{O}$-Notation, Rekursion, Datenstrukturen
- Sortieren
- Wörterbücher und Mengen
- Graphen und Graphalgorithmen

In der vorigen (und auch in dieser) Vorlesung werden wir erfahren,

1. wie man andere Mengenoperationen effizient mit (fast) ausgeglichenen Suchbäumen implementiert,

genauer werden wir hierzu wieder AVL-Bäume heranziehen

2. wie man Mengen mit anderen “Bedürfnissen” verwaltet, z.B.:

2a. Prioritätswarteschlangen oder

2b. Äquivalenzklassen

Außerdem erwartet uns eine Einführung in Graphalgorithmen.

Partitionen

Es sei M eine Menge, $M \neq \emptyset$. Eine *Zerlegung*, *Klasseneinteilung* oder *Partition* von M ist eine *Mengenfamilie* $Z \subseteq 2^M$ mit:

1. $M = \bigcup_{A \in Z} A$.

2. $\emptyset \notin Z$.

3. $\forall A, B \in Z : A \cap B \neq \emptyset \implies A = B$.

Die Elemente von Z heißen auch *Klassen*.

Partitionen und Äquivalenzrelationen

Lemma: Eine Klasseneinteilung Z von M induziert eine Äquivalenzrelation $\sim_Z$ über M durch $a \sim_Z b \iff a$ und b liegen in derselben Z-Klasse.

Lemma: Ist R eine Äquivalenzrelation über M, so ist

$$Z_R = \left\{ \underbrace{\{a \in M \mid aRb\}}_{=:[b]_R} \mid b \in M \right\}$$

eine Zerlegung von M, die so genannte *Quotientenmenge*, oft geschrieben M/R. b heißt auch *Repräsentant* der *Äquivalenzklasse* $[b]_R$.

Genauer gilt: Lemma: Ist R ÄR, so gilt $R = (\sim_{Z_R})$.

Partitionen und Äquivalenzrelationen: Beispiele

Beispiel: Allrelation: $[x]_{M\times M} = M$ für alle $x \in M$.

Beispiel: Diagonale: $[x]_{\Delta_M} = \{x\}$ für alle $x \in M$.

Beispiel: $R_m \subset \mathbb{Z} \times \mathbb{Z}$: Äquivalenzklassen sind $[0]$, $[1]$, $\ldots$, $[m-1]$.

Hinweis: Zwei ganze Zahlen sind äquivalent gdw. sie lassen beim Teilen durch m denselben Rest. Schreibweise: $\mathbb{Z}_m := \mathbb{Z}/R_m$.

Partitionen und Äquivalenzrelationen: $\mathbb{Z}$ und $\mathbb{Q}$ als Beispiele

Man kann $\mathbb{N}$ *axiomatisch* eingeführen (oder Kronecker folgen).
Daraus könnte man $\mathbb{Z}$ einführen als Quotientenmenge auf $\{+,-\} \times \mathbb{N}$ mit der Äquivalenzrelation $(v, n) \sim (v', n')$ gdw. $n = n' \wedge (v = v' \vee n = 0)$.

Über $M = \mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ definiere: $(a, b) \sim (a', b') \iff ab' = ba'$. Dann ist die Menge der rationalen Zahlen $\mathbb{Q}$ definiert als $\mathbb{Q} = M/\sim$.
Statt $[(a, b)] \in M/\sim$ schreiben wir auch $\frac{a}{b}$.
(a, b) und (a', b') liegen in derselben Äquivalenzklasse gdw. $\frac{a}{b} = \frac{a'}{b'}$ gdw. $ab' = ba'$.

Operationen auf Zerlegungen

Gegeben sei eine Partition einer Menge S und eine Folge von Äquivalenzanweisungen

$$a_1 \equiv b_1, a_2 \equiv b_2, \ldots, a_m \equiv b_m.$$

Man verschmelze bei jeder Anweisung $a_i \equiv b_i$ die Äquivalenzklassen von a_i und b_i. Das entspricht einer *disjunkten Vereinigung*, engl. “Union” oder “Merge”. Außerdem soll die Äquivalenklasse von $s \in S$ effizient ermittelt werden, engl. “Find”.

Erste Lösung: zwei Felder

IST_IN[0..N-1] enthält dabei den Namen der Menge, die i enthält.
UNION$(\mathrm{A}, \mathrm{B}, \mathrm{C})$ (d.h.: Vereinige die disjunkten Mengen A und B zu dem Ergebnis C) ist jetzt zwar einfach zu implementieren, aber langsam (linear):
Man laufe über das Feld IST_IN und ändere die Mengenbezeichnungen A bzw. B zu C.
Beschleunigungsidee: (a) Laufe nur über die Elemente von A und B; (b) Laufe nur über die kleinere der Mengen A oder B durch Unterscheidung von internen und externen Mengennamen (Felder MAPIN bzw. MAPOUT).

Ein besserer feldbasierter Algorithmus für Union/Find

Input(s): Zu vereinigende Mengen A und B.

Output(s): Struktur, in der A und B vereinigt wurden.

if $|A| \leq |B|$ **then**
  $X \leftarrow A$; $Y \leftarrow B$
else
  $X \leftarrow B$; $Y \leftarrow A$
$j \leftarrow \text{MAPIN}[Y]$
for all $x \in X$ **do**
  IST_IN$[x] \leftarrow j$
Vereinige die Listen MAPIN$[X]$ und j und nenne die entstehende Liste j.
MAPOUT$[j] \leftarrow C$; MAPIN$[C] \leftarrow j$

Satz: Mit dem soeben gezeigten Programm kann man eine Folge von $n - 1$ Union-Operationen und m Find-Operationen in Zeit $\mathcal{O}(m + n \log(n))$ abarbeiten.

Beweis: Ein einzelnes Find kostet $\mathcal{O}(1)$ Zeit.

Union(A, B, C) kostet $\mathcal{O}(\min\{|A|, |B|\})$, d.h. höchstens $c \cdot \min\{|A|, |B|\}$.

Trick: *amortisierte Analyse*: Jedem Element von X (der kleineren der beiden Mengen A bzw. B ordnen wir genau c Zeiteinheiten zu.

Behauptung: Jedem $x \in U$ (Universum) werden höchstens $c \log_2(n)$ Zeiteinheiten zugeordnet.

Wird x für die Teilnahme an einem Union etwas zugeordnet, dann liegt x nach diesem Union in einer Menge, die mindestens doppelt so groß ist wie zuvor. Wir beginnen mit lauter einelementigen Mengen und können daher mit $n - 1$ Union-Operationen höchstens Mengen der Größe n aufbauen, d.h., x werden höchstens $\log_2(n)$-mal c Zeiteinheiten zugeordnet.

Aus der Zwischenbehauptung ergibt sich der Satz, denn höchstens $2(n - 1)$ Elemente aus U nehmen überhaupt an den $n - 1$ Unions teil.

Zweite Lösung: Bäume

Mit Bäumen (knotenorientierte Mengenspeicherung) lässt sich Union einfach implementieren.
(Zusätzlich speichert die Wurzel den Mengennamen.)
Find ist jetzt allerdings teuer; bei logarithmischer Baumhöhe kostet diese Operation nun logarithmische Zeit, falls die Baumknoten konstanten Grad besitzen.
Alternative Abschätzung in n möglich, da höchstens $n - 1$ Union-Operationen ausgeführt wurden.
Um beim Union "schiefe Bäume" zu vermeiden, hilft eine einfache *gewichtete Vereinigungsregel*:
Union(A, B, C) bedeutet: Mache die Wurzel des Baumes für die kleinere Menge zu einem (weiteren) Sohn der Wurzel des Baumes für die größere Menge (und speichere den Namen C an der Wurzel).

Sei $\rho(x)$ der Rang von x, also die Höhe des bei x beginnenden Baumes, und $\gamma(x)$ das *Gewicht* von x, d.h., die Zahl der Nachkommen von x.

Lemma: Wird die gewichtete Vereinigungsregel benutzt, so gilt $\forall x \in U : \gamma(x) \geq 2^{\rho(x)}$.

Beweis: Induktion über $\rho(x)$. ✓ für $\rho(x) = 0$.

Gilt $\rho(x) \geq 1$, so hat x einen Sohn y mit $\rho(y) = \rho(x) - 1$.

Als y zum Sohn von x wurde, hatte x mindestens soviele Nachkommen wie y.

Nach dem Union war die Zahl der Nachkommen von x mindestens doppelt so groß wie der von y.

Spätere Union-Operationen geben y keine neuen Nachkommen, sodass nach Induktionsannahme gilt: $\gamma(x) \geq 2\gamma(y) \geq 2 \cdot 2^{\rho(y)} = 2^{\rho(x)}$.

Außerdem gilt (da wir mit Einermengen starten): $\forall x \in U : \gamma(x) \leq n$.

Satz: Die gewichtete Vereinigungsregel gestattet eine Folge von $n - 1$ Union-Operationen und m Find-Operationen in Zeit $\mathcal{O}(n + m \log(n))$ abzuarbeiten.

Die Ackermann-Funktion und ihre Inverse

siehe auch VL über “Berechenbarkeit und Komplexität” bzw. GTI-2 oder GTI-3.

Definiere $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ durch:

$A(i, 0) = 1$, für $i \geq 1$

$A(0, x) = 2x$, für $x \geq 0$

$A(i + 1, x + 1) = A(i, A(i + 1, x))$ für $i, x \geq 0$.

Dies ist die berühmte (nicht primitiv-rekursive) *Ackermann-Funktion*.

Ihr sogenanntes Inverses kann man wie folgt beschreiben:

$\alpha(m, n) = \min\{z \geq 1 \mid A(z, 4\lceil m/n \rceil) > \log(n)\}$.

Während A extrem schnell wächst, wächst α extrem langsam, d.h., $\alpha(m, n)$ ist konstant für alle praktischen Werte von m, n.

Pfadkomprimierung

Grundsatz: Wann immer ein Find ausgeführt wird, indem man die Knoten, die man auf der Suche nach x besuchen musste, zu unmittelbaren Söhnen der Wurzel macht. Das reduziert die Suchkosten für künftige Finds.

Satz: Wenn Pfadkomprimierung und die gewichtete Vereinigungsregel benutzt wird, hat eine Folge von $n - 1$ Unions und $m \geq n$ Finds die Zeitkomplexität $\mathcal{O}(m\alpha(m, n))$.

Der Beweis findet sich z.B. auf meheren Seiten im Buch von Mehlhorn (und entfällt hier...)

Ein Pfad-Problem als Hinführung zu Graphenalgorithmen

KÜRZESTE PFADE IN GRAPHEN

Eingabe: gerichteter Graph $G = (V, A)$,
Kantenlängen $\ell : A \to \mathbb{R}_{\geq 0}$,
Ausgangsknoten $s \in V$

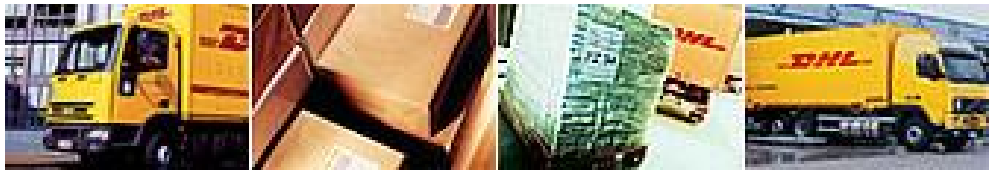
Frage: Längen der kürzesten Pfade von s zu allen anderen Knoten

Anwendungsbeispiel: Lieferrouten

G: modelliert das Wegenetz und damit auch die Auslieferungsorte

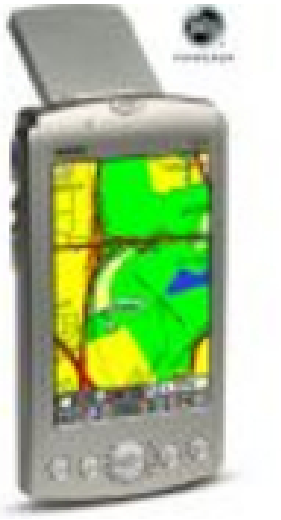
ℓ: Abstand (räumlich oder zeitlich)

s: Standort des Lagers

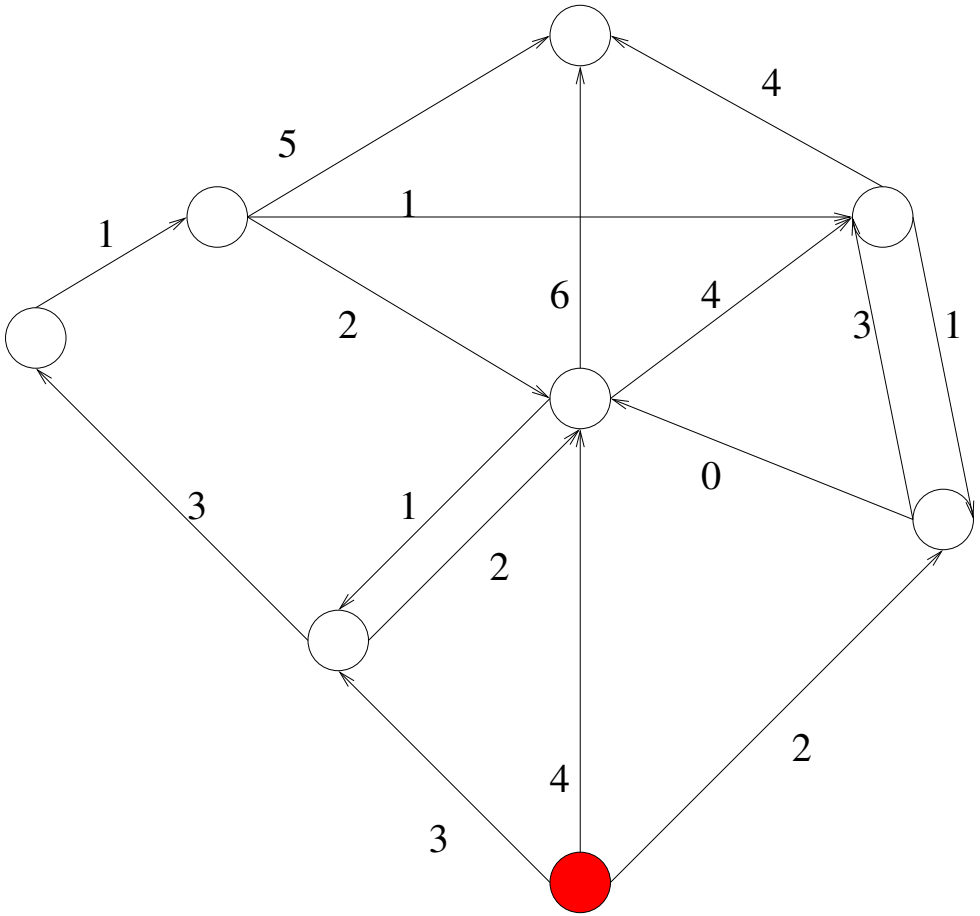


Frage: Wie kommt die Ware am besten zum Kunden?

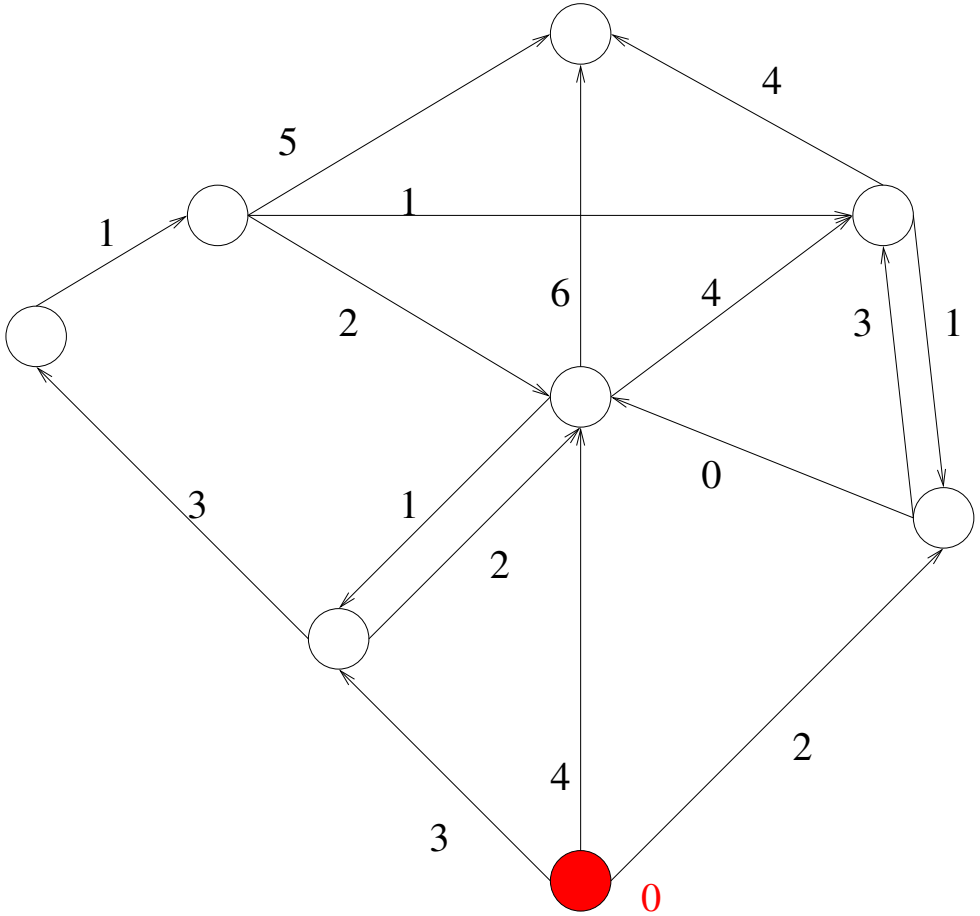
Etwas individueller: Navigationssysteme



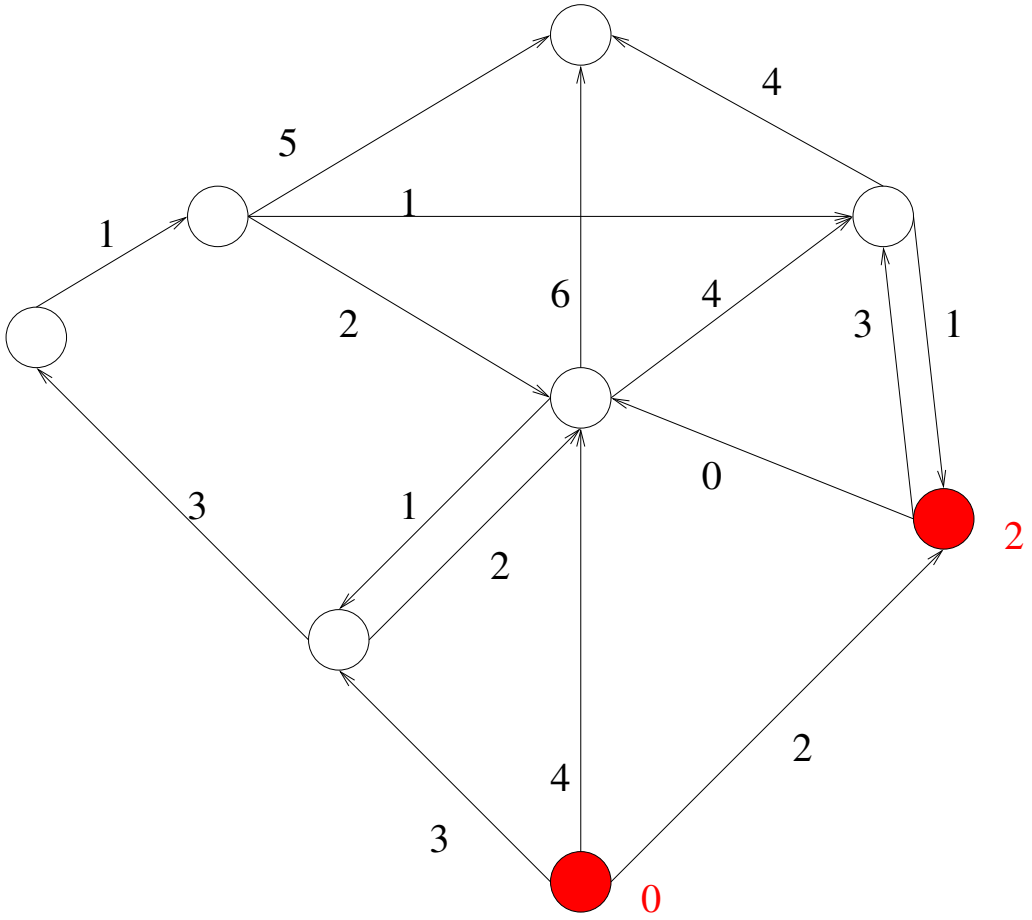
Ein etwas abstrakteres Beispiel



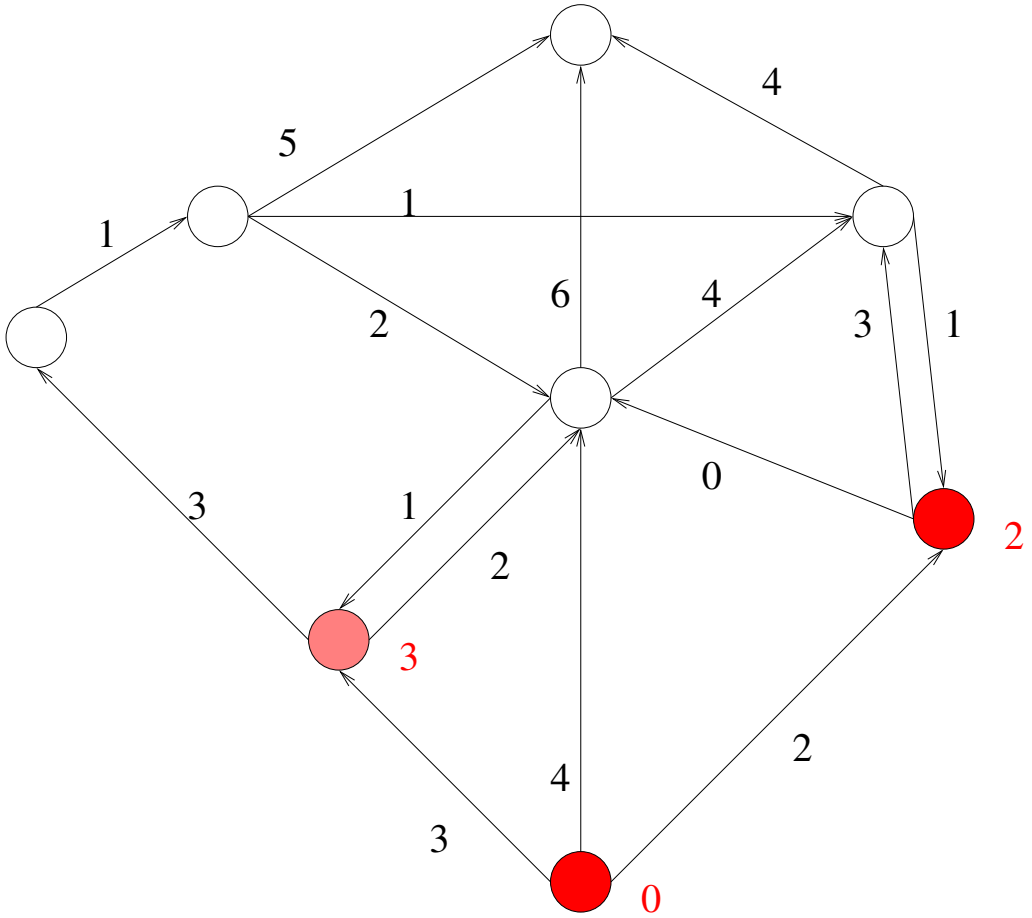
Die Menge S: Abstände zu s sind bekannt!



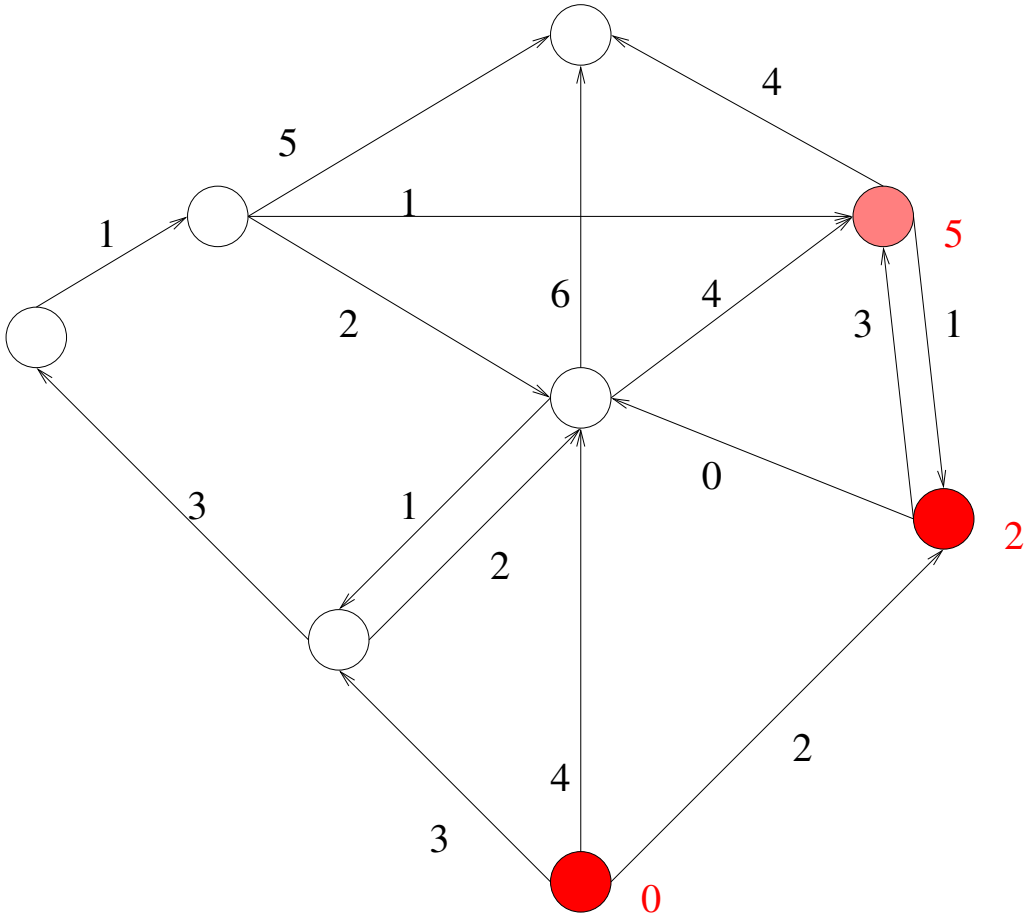
Die Menge S: $|S| = 2$



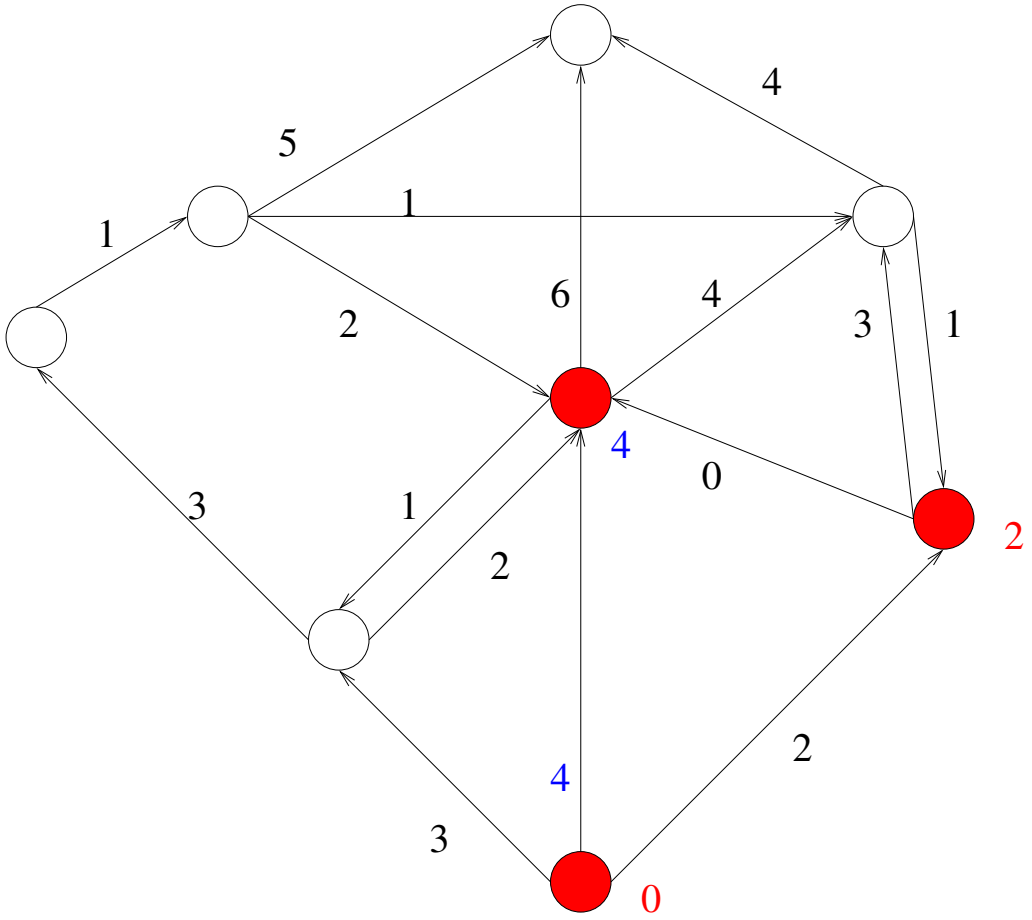
Die Menge S: $|S| = 3$, 1. Versuch



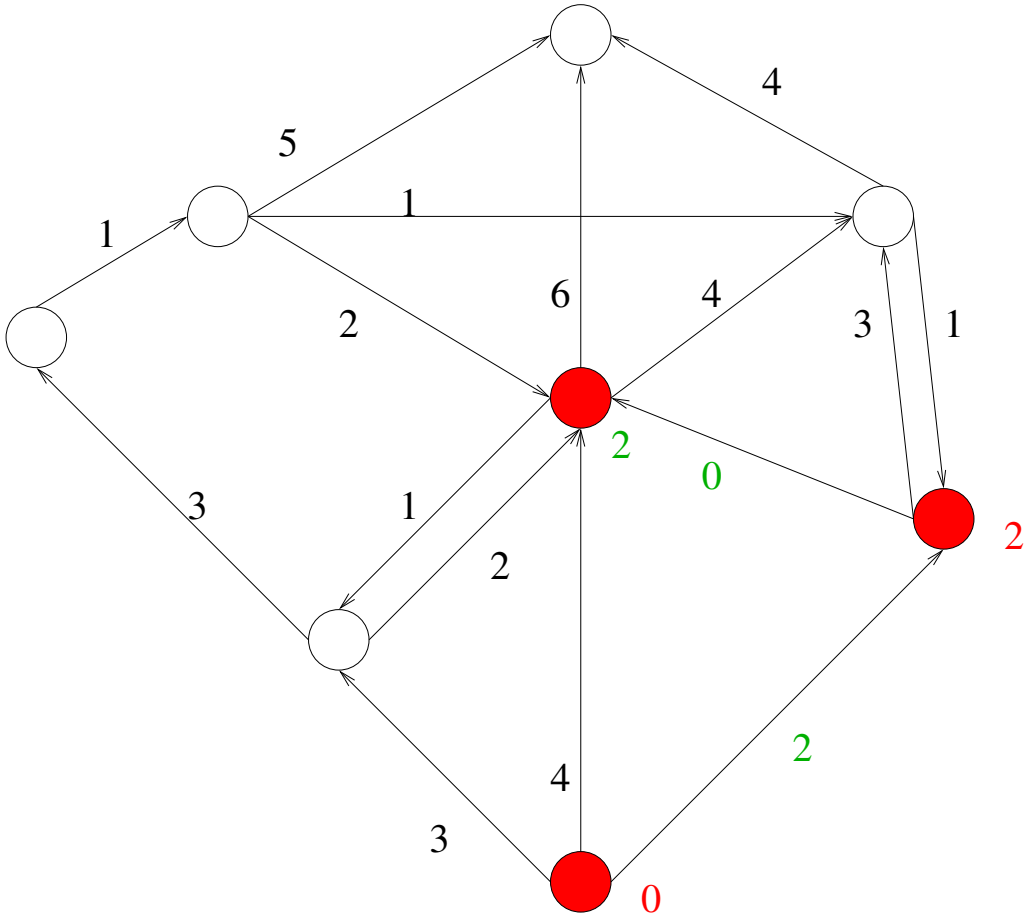
Die Menge S: $|S| = 3$, 2. Versuch



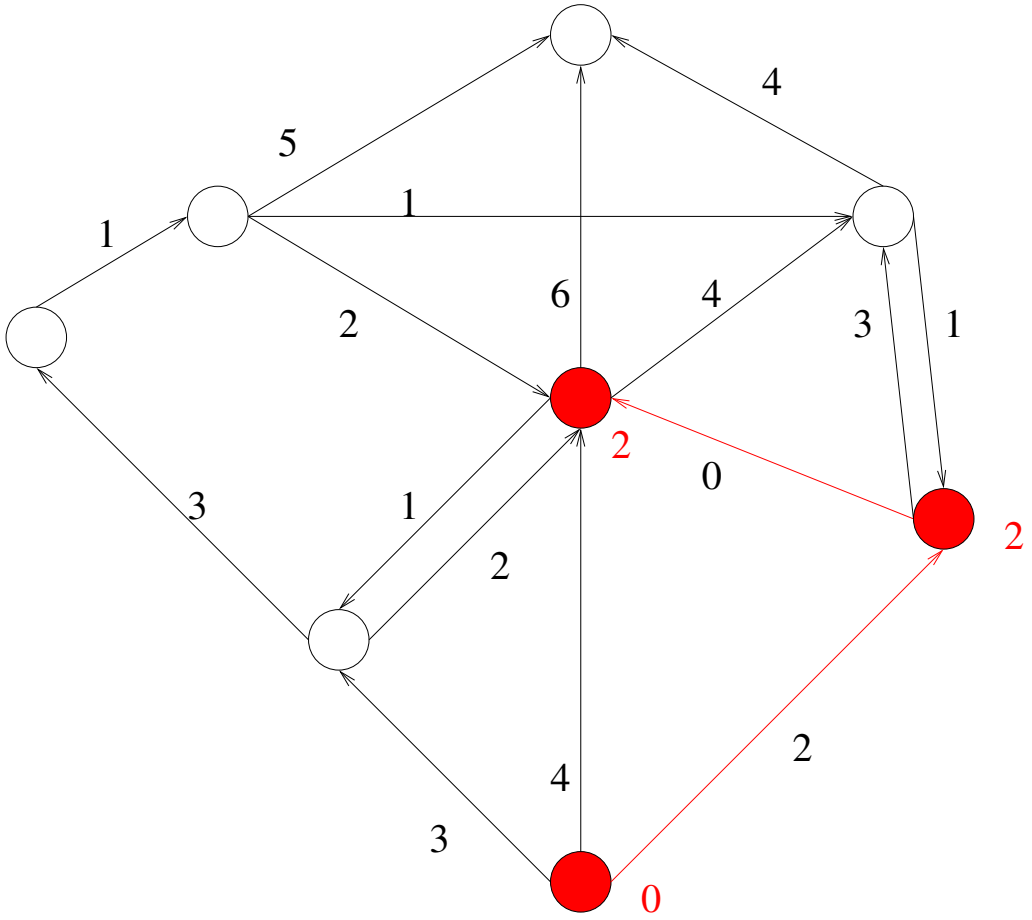
Die Menge S: $|S| = 3$, 3. Versuch, 1. Möglichkeit



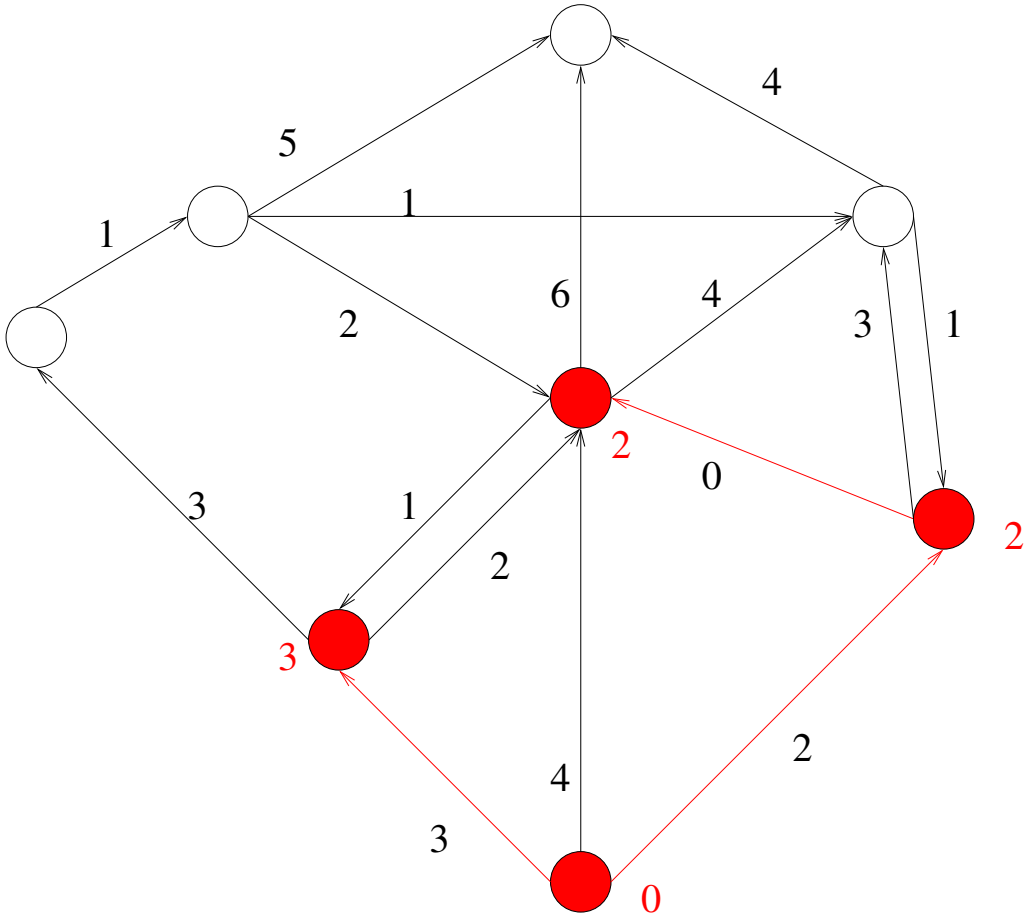
Die Menge S: $|S| = 3$, 3. Versuch, 2. Möglichkeit



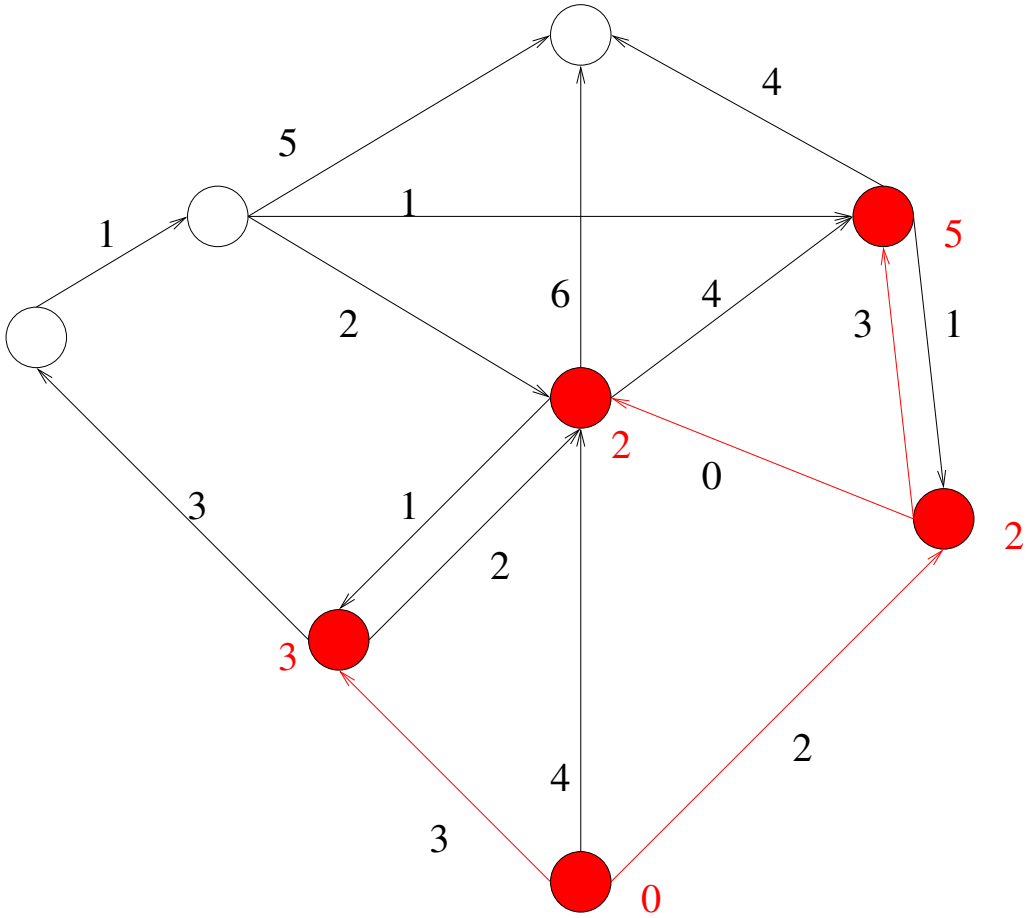
Die Menge S: $|S| = 3$



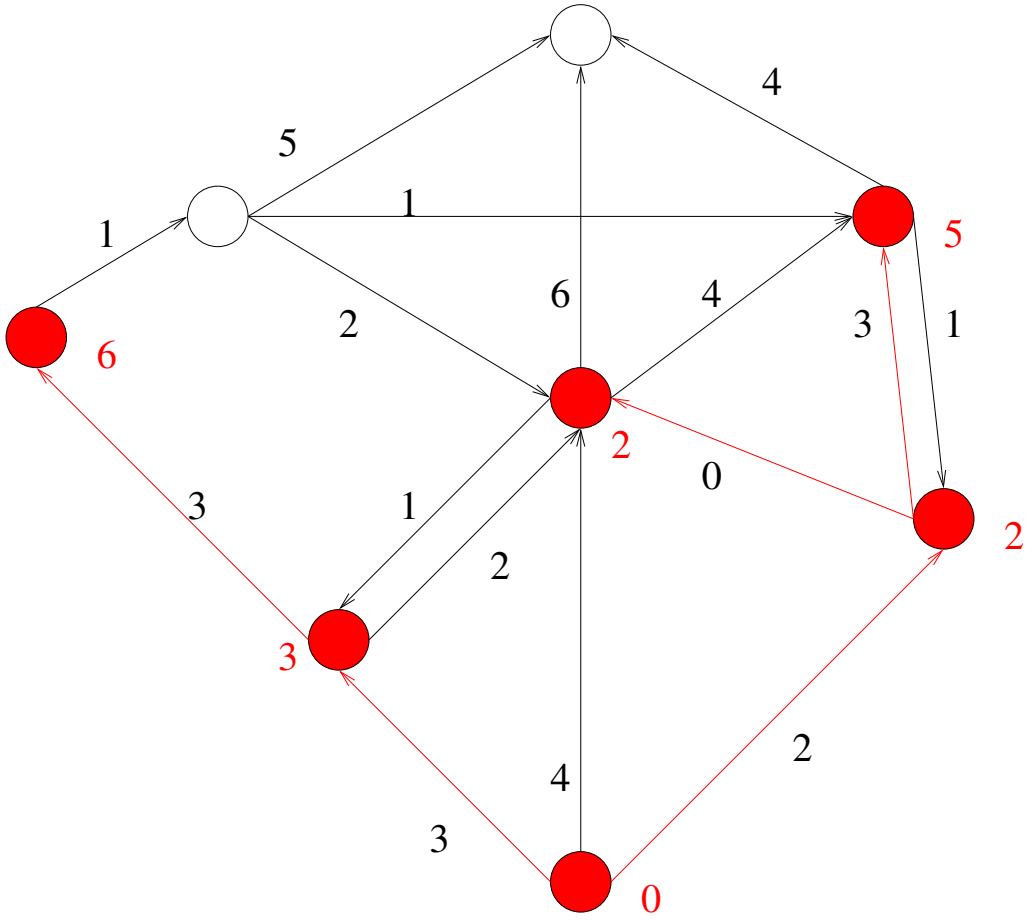
Die Menge S: $|S| = 4$



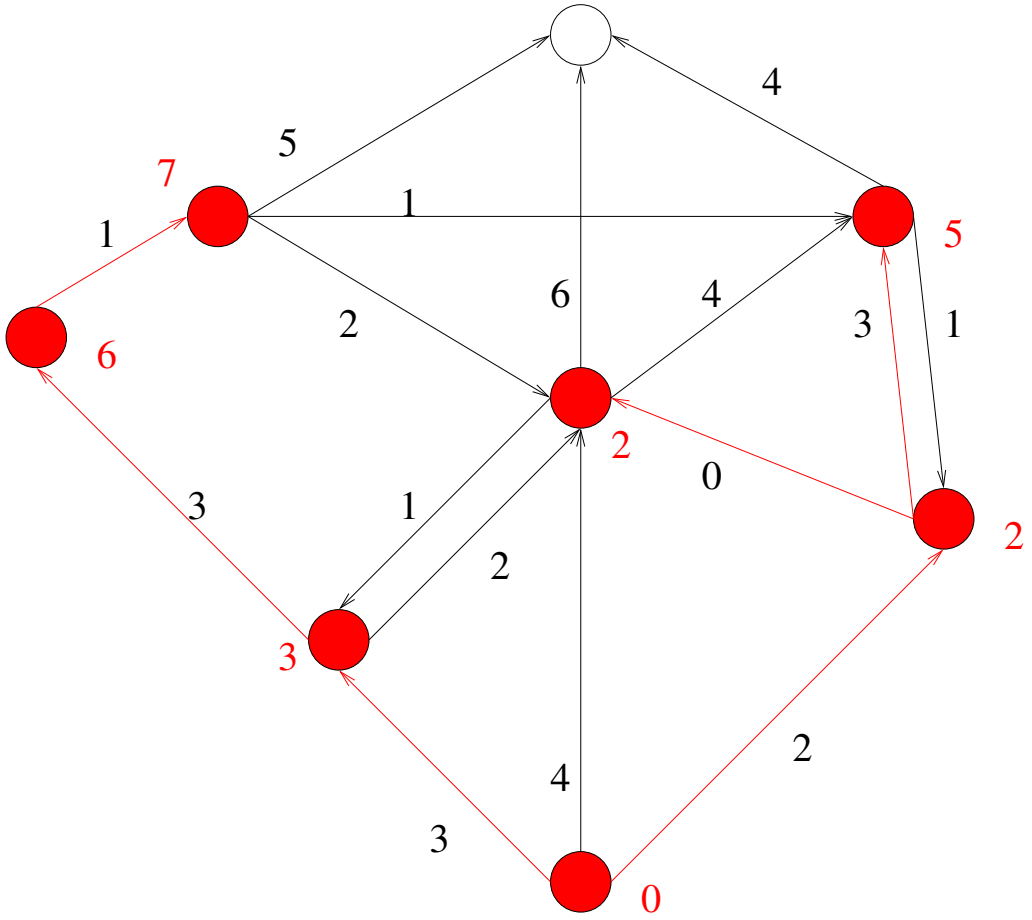
Die Menge S: $|S| = 5$



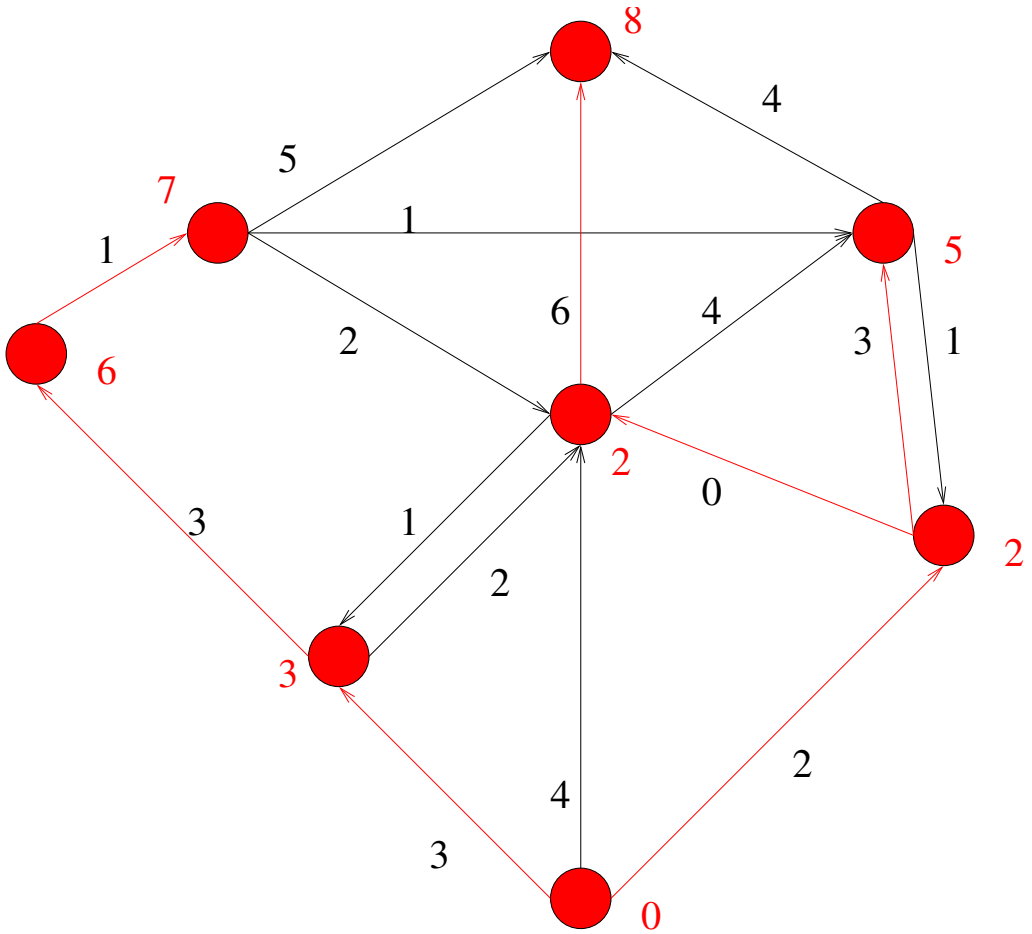
Die Menge S: $|S| = 6$



Die Menge S: $|S| = 7$



Die Menge S: $|S| = 8$



DIJKSTRA: Kürzeste Pfade in Graphen

Input(s): Gerichteter Graph $G = (V, A)$ mit Kantenlängenfunktion ℓ und Ausgangsknoten s, Menge $S \subseteq V$ ($s \in S$) und Abstandsfunktion d, auf S definiert.

Output(s): d: enthält die Längen kürzester Pfade von s nach $t \in S$

if $S = V$ **then**

d enthält alle gewünschten Abstände $\rightsquigarrow$ fertig!

else

Wähle $t \in V \setminus S$ so, dass $d'(t) = \min\{d(v) + \ell(\vec{vt}) \mid v \in S\}$ minimal.

Erweitere S um t und setze $d(t) = d'(t)$.

Berechne DIJKSTRA(G, ℓ, s, S, d)

Warum gilt: d enthält die Längen kürzester Pfade von s nach $t \in S$?

Initialisierung (Induktionsanfang): $S = \{s\}$, $d(s) = 0$. ✓

Induktionsschritt:
Annahme: DIJKSTRA will t zu S hinzufügen, aber $d'(t)$ ist größer als der wahre Abstand $d(t)$ von s zu t.
Nach Induktion: Vorgänger von t auf Minimalpfad P liegt nicht in S.
Sei x der erste Knoten auf P, der nicht in S liegt.
$d'(x) \leq d(t) < d'(t)$ widerspricht nun der Wahl von t.

Zur Laufzeit

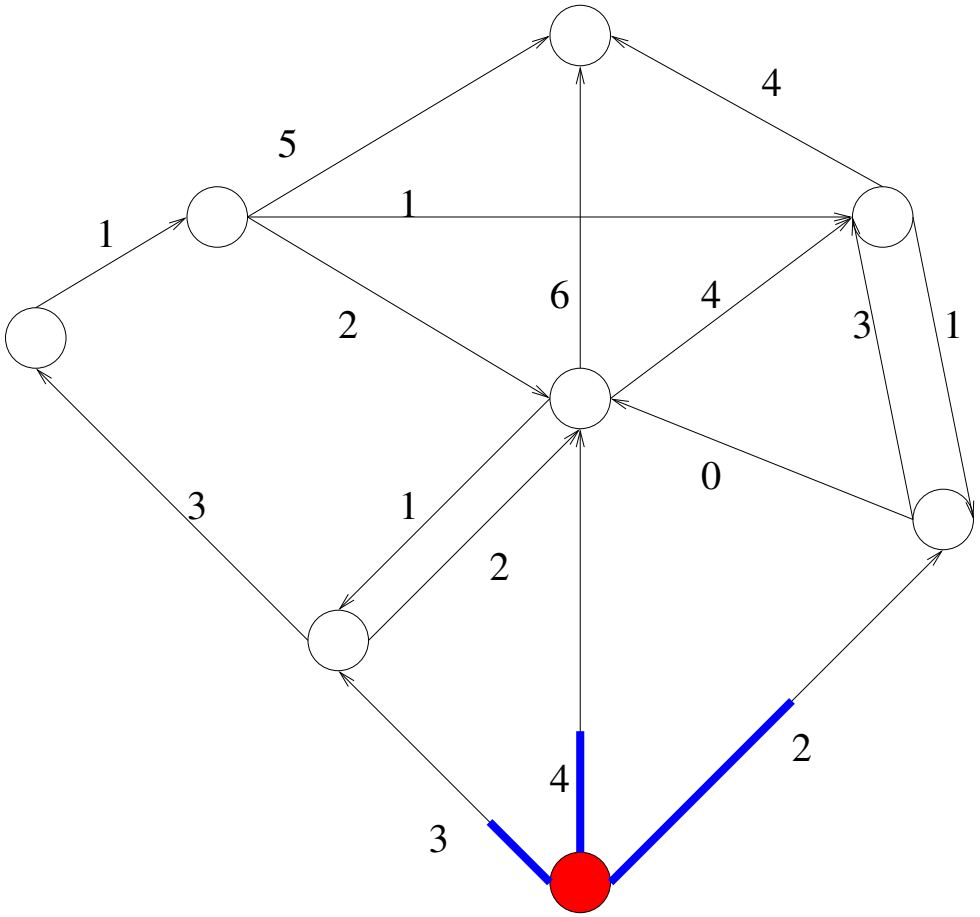
Polynomiell ✓

Durch geeignete Datenstrukturen auf $O(m \log n)$ verbesserbar.

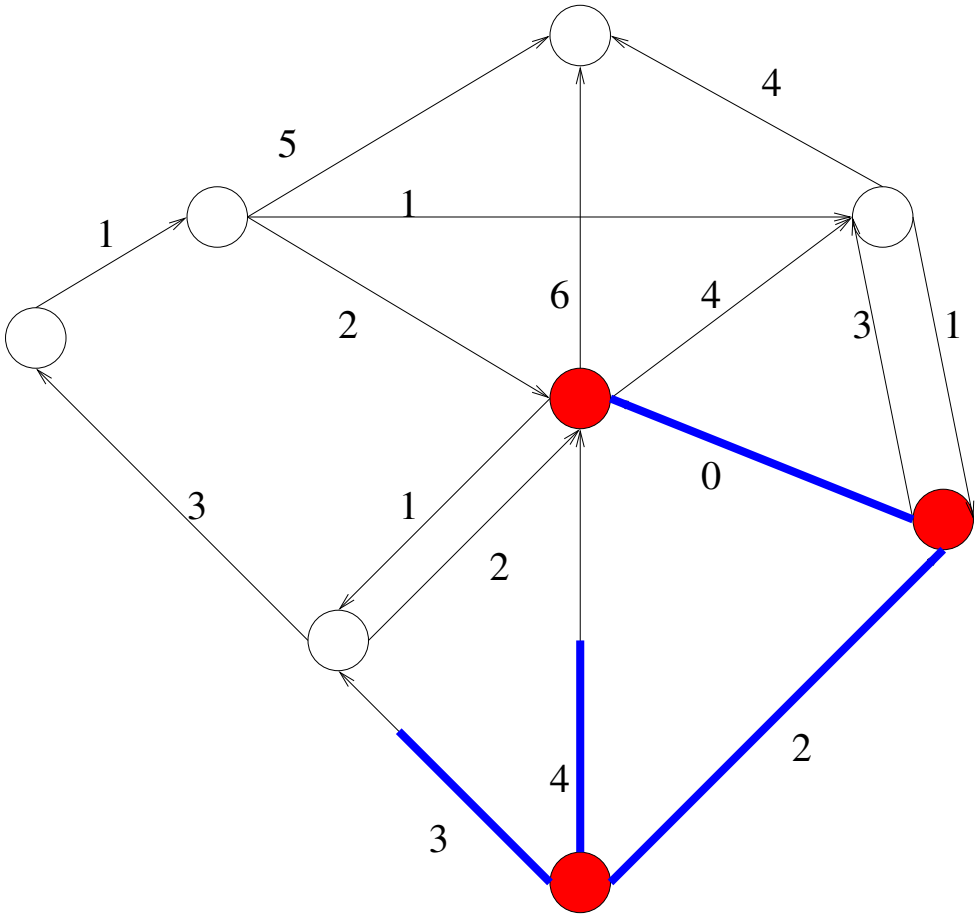
Weitere Kommentare

- Spannbaumkonstruktion
- DIJKSTRA als zeitdiskrete Breitensuche
- Wassermodell

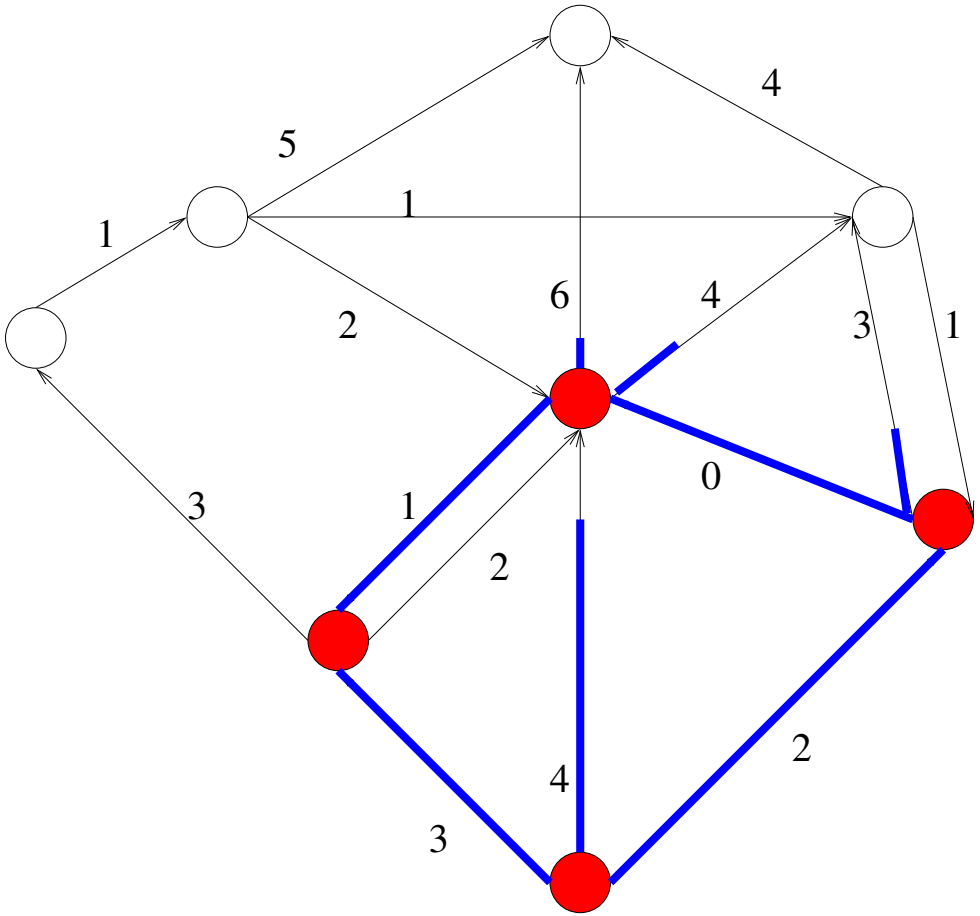
Zeitpunkt 1



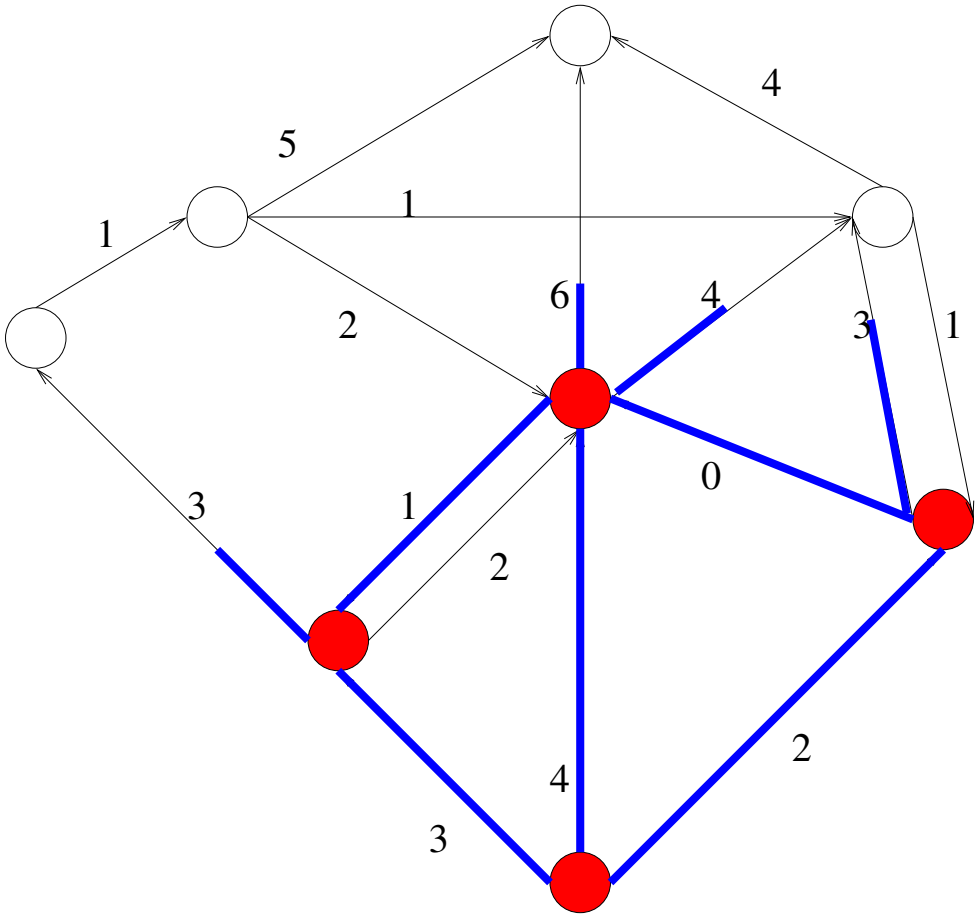
Zeitpunkt 2



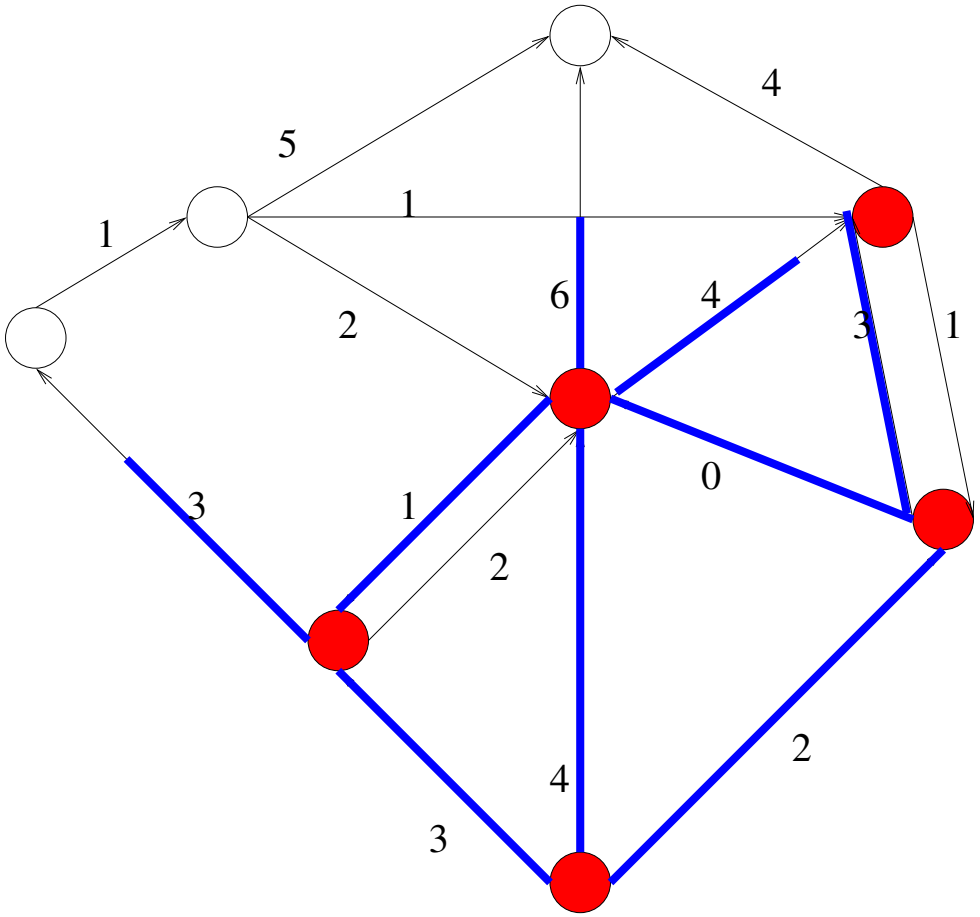
Zeitpunkt 3



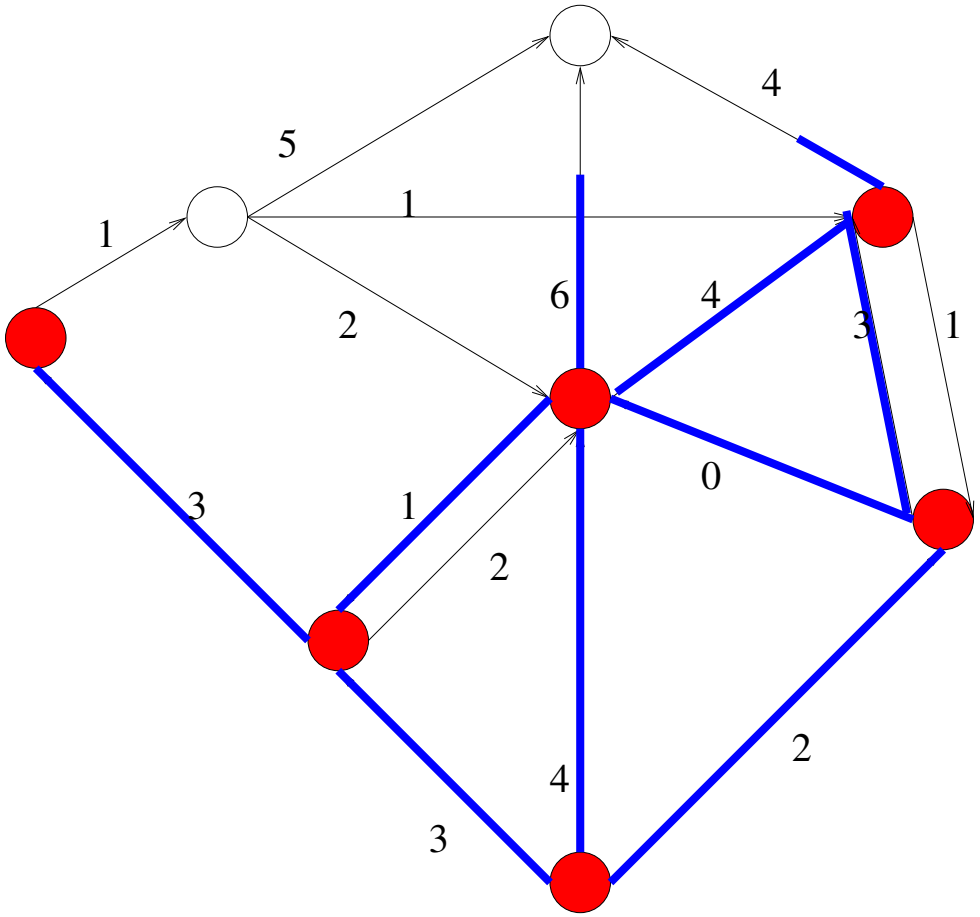
Zeitpunkt 4



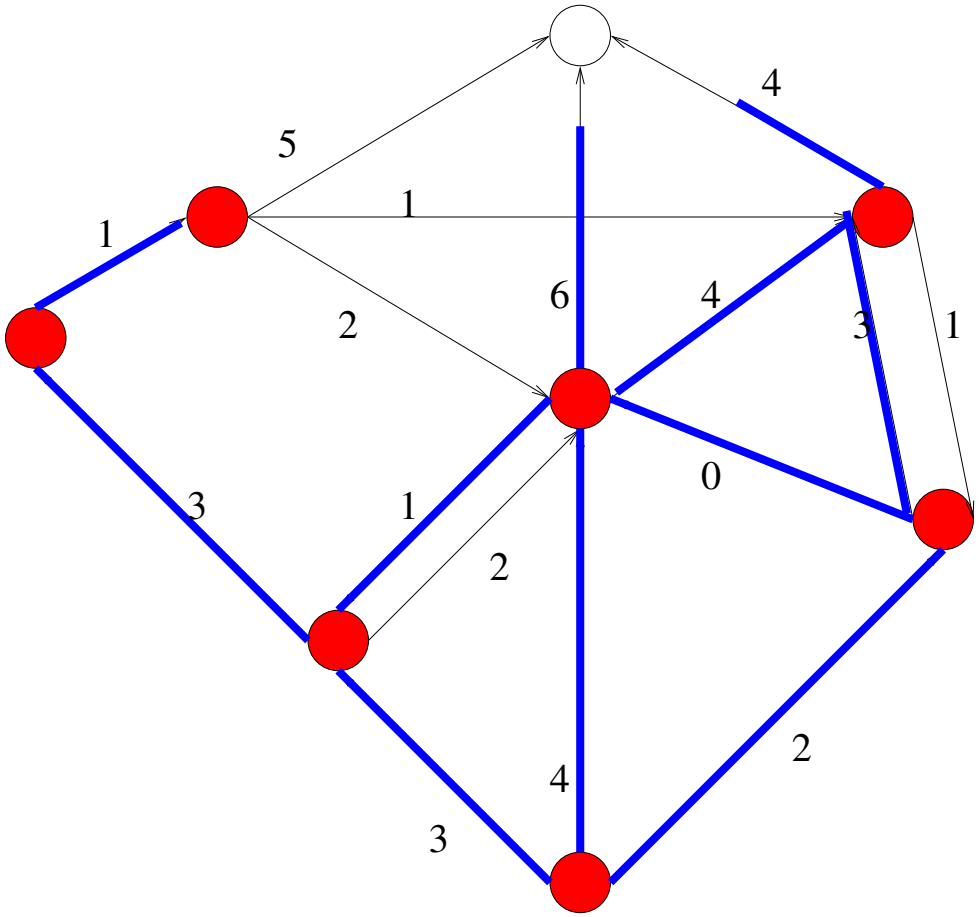
Zeitpunkt 5



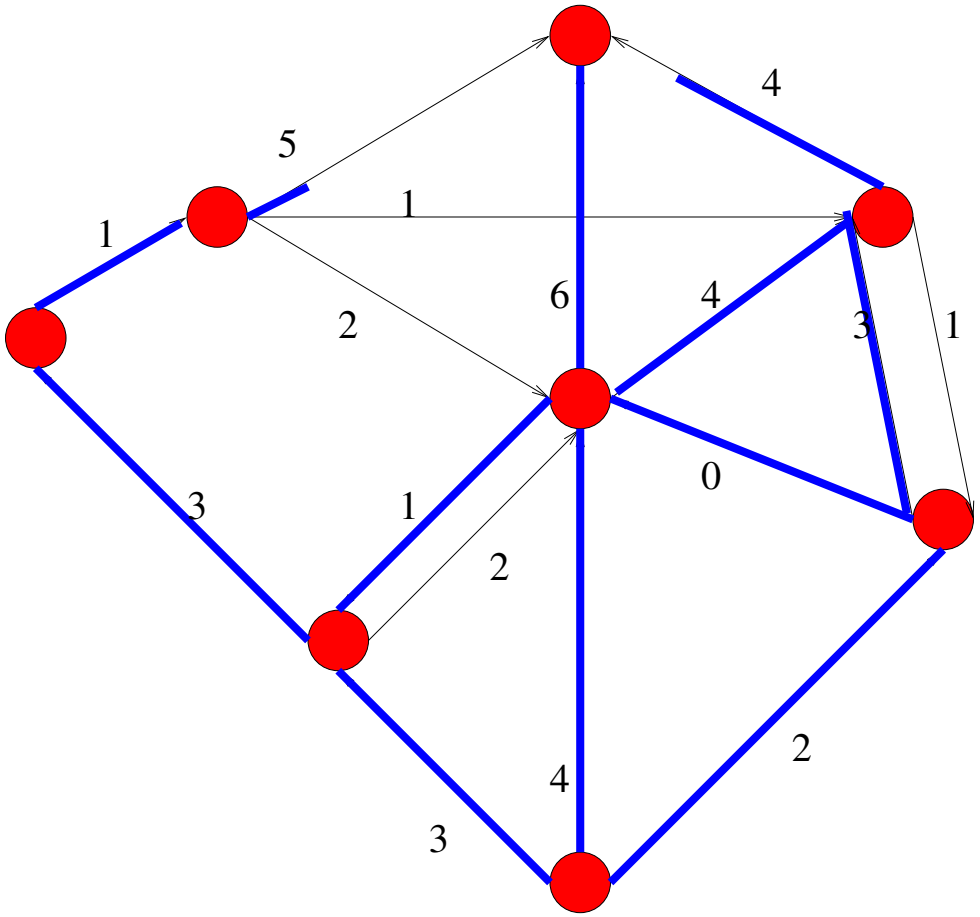
Zeitpunkt 6



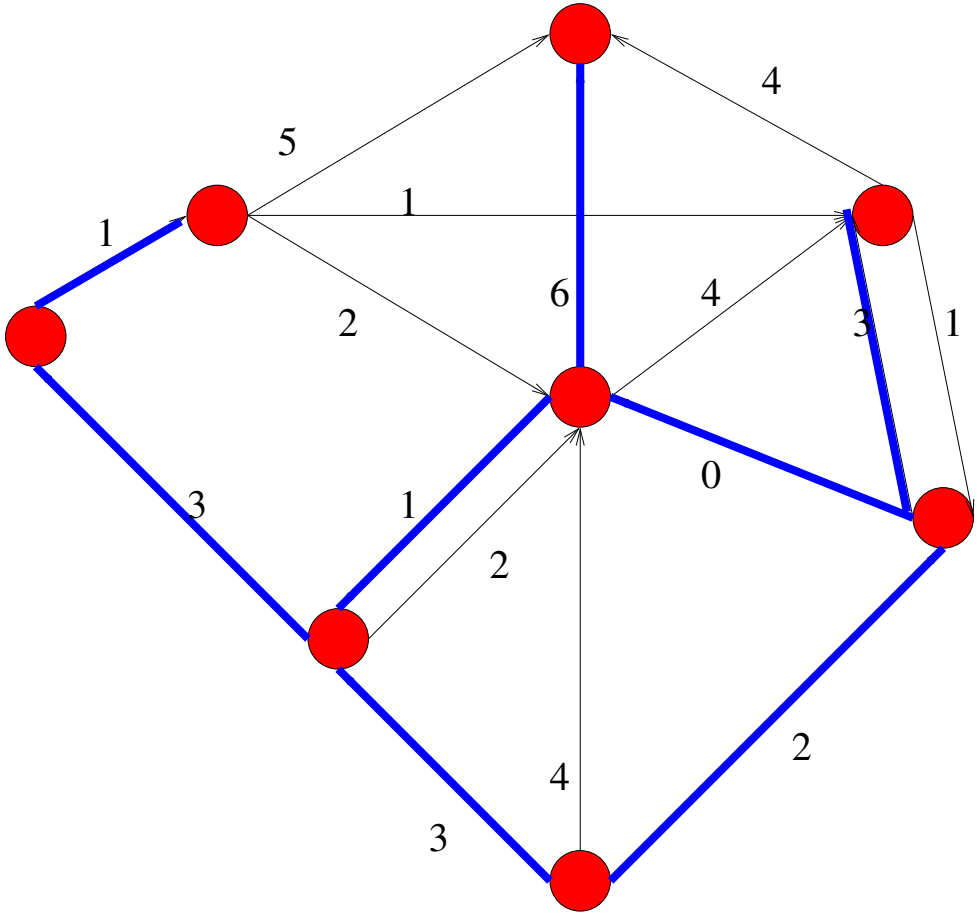
Zeitpunkt 7



Zeitpunkt 8



Zeitpunkt 8 (mit “abgebrochenen Fehlversuchen”)



Eine schöne Animation von Siena

file:
///home/fernau/tex/lehre/Algorithmen/SoSe08/dijkstra.html