ADS: Algorithmen und Datenstrukturen 2

Teil 8

Prof. Dr. Gerhard Heyer

Institut für Informatik
Abteilung Automatische Sprachverarbeitung
Universität Leipzig

30. Mai 2018

[Letzte Aktualisierung: 02/07/2018, 06:45]

Kürzeste Wege

- kantenmarkierter (gewichteter) Graph $G = (V, E, g)$
  - Weg/Pfad P der Länge n: $(v_0, v_1), (v_1, v_2), , ..., (v_{n-1}, v_n)$
  - Gewicht (Länge) des Weges/Pfads $w(P) = \sum_{i=1}^{n} g(v_{i-1}, v_i)$
  - Distanz $d(u, v)$: Gewicht des kürzesten Pfades von u nach v
- Varianten
  - nicht-negative Gewichte vs. negative und positive Gewichte
  - Bestimmung der kürzesten Wege:
    a) zwischen allen Knotenpaaren,
    b) von einem Knoten u aus
    c) zwischen zwei Knoten u und v
- Bemerkungen
  - kürzeste Wege sind nicht immer eindeutig
  - kürzeste Wege müssen nicht existieren, z.B. wenn:
    - kein Weg existiert oder
    - ein Zyklus mit negativem Gewicht existiert

Zur Erinnerung: Warshall's Transitive Hülle

$A =$ Adjazenzmatrix $G(V, E)$, $n = |V|$

```
boolean[][] A= {...}; //Adjazenzmatrix

for (int k=1; k<=A.length; k++)
  for (int i=1; i<=A.length; i++)
    if (A[i][k])
      for (int j=1; j<=A.length; j++)
        if (A[k][j]) A[i][j]=true;
```

Warshall's Algorithmus für Distanzen

Warshall-Algorithmus lässt sich einfach modifizieren, um kürzeste Wege zwischen allen Knotenpaaren zu berechnen

```
Fuer alle Kanten i,j setze A[i][j] = g(i,j), g[i,i] = 0 und
g[i,j] = unendlich, falls keine Kante zwischen i und j
```

```
Von k = 1 bis n
   Von i = 1 bis n
      Von j = 1 bis n
         A[i][j] = min (A[i][j], A[i][k] + A[k][j])
```

Annahme: kein Zyklus mit negativem Gewicht vorhanden
Komplexität: $O(n^3), n = |V|$

Dijkstra-Algorithmus I

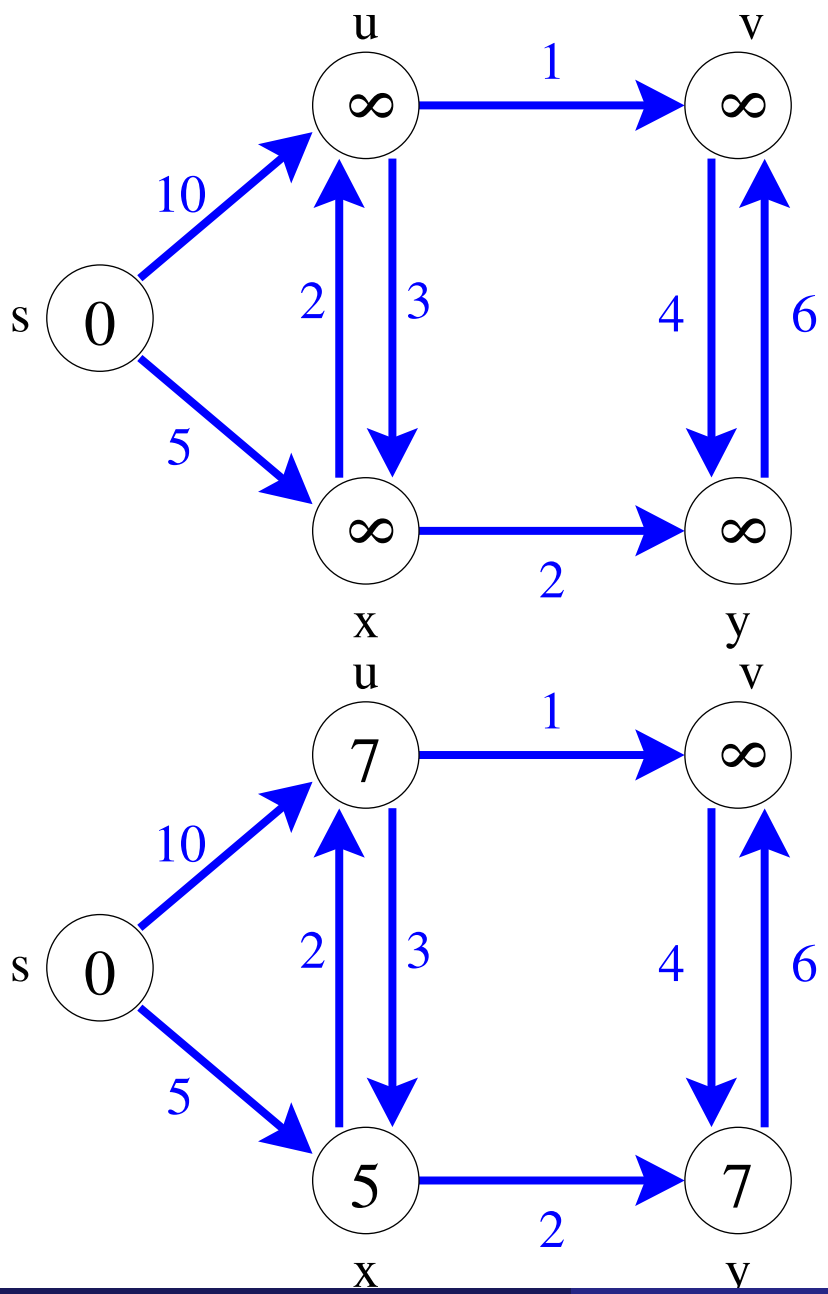
Bestimmung der von einem Knoten ausgehenden kürzesten Wege

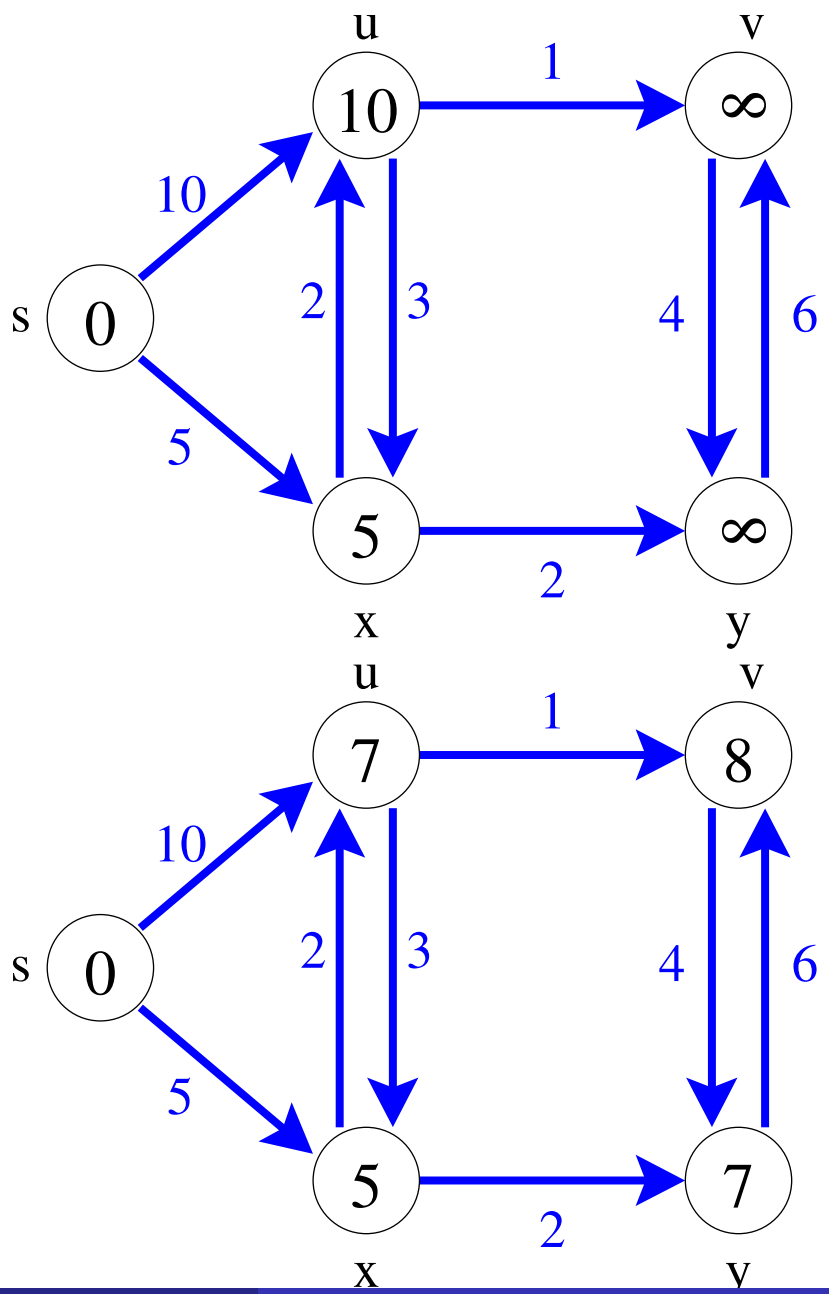
- gegeben: kanten-bewerteter Graph $G = (V, E, g)$ mit $g : E \to \mathbb{R}_0^+$ (Kantengewichte)
- Startknoten s; zu jedem Knoten u wird die Distanz zu Startknoten s in $D[u]$ geführt
- Q sei Prioritäts-Warteschlange (sortierte Liste);
  Priorität = Distanzwert
- Funktion $succ(u)$ liefert die Menge der direkten Nachfolger von u
- Verallgemeinerung der Breitensuche (gewichtete Entfernung)
- funktioniert nur bei nicht-negativen Gewichten
- Optimierung gemäß Greedy-Prinzip

Dijkstra-Algorithmus II

```
Fuer alle Knoten v setze D[v] = unendlich
D[s] = 0
PriorityQueue Q = V
Solange Q nicht leer dann
   v = naechster Knoten aus Q mit kleinstem Abstand D
   Entferne v aus Q
   Fuer jeden Nachbarn u in succ(v) & Q
      Falls D[v] + g(v,u) < D[u] dann
         D[u] = D[v] + g(v,u)
```

Dijkstra: Beispiel





Dijkstra-Algorithmus: Korrektheit

Korrektheitsbeweis

- nach i Schleifendurchgängen sind die Längen von i Knoten, die am nächsten an s liegen, korrekt berechnet und diese Knoten sind aus Q entfernt.
- Induktionsanfang: s wird gewählt, $D(s) = 0$
- Induktionsschritt: Nimm an, v wird aus Q genommen. Der kürzeste Pfad zu v gehe über direkten Vorgänger v' von v. Da v' näher an s liegt, ist v' nach Induktionsvoraussetzung mit richtiger Länge $D(v')$ bereits entfernt. Da der kürzeste Weg zu v die Länge $D(v') + g(v', v)$ hat und dieser Wert bei Entfernen von v' bereits v zugewiesen wurde, wird v mit der richtigen Länge entfernt.
- erfordert nicht-negative Kantengewichte (wachsende Weglänge durch hinzugenommene Kanten)

Dijkstra-Algorithmus: Eigenschaften

Komplexität $O(n^2), n = |V|$

- n-maliges Durchlaufen der äußeren Schleife liefert Faktor $O(n)$
- innere Schleife: Auffinden des Minimums begrenzt durch $O(n)$, ebenso das Aufsuchen der Nachbarn von v

Bestimmung des kürzesten Weges zwischen u und v:
Spezialfall für Dijkstra-Algorithmus mit Start-Knoten u (Beendigung sobald v aus Q entfernt wird)

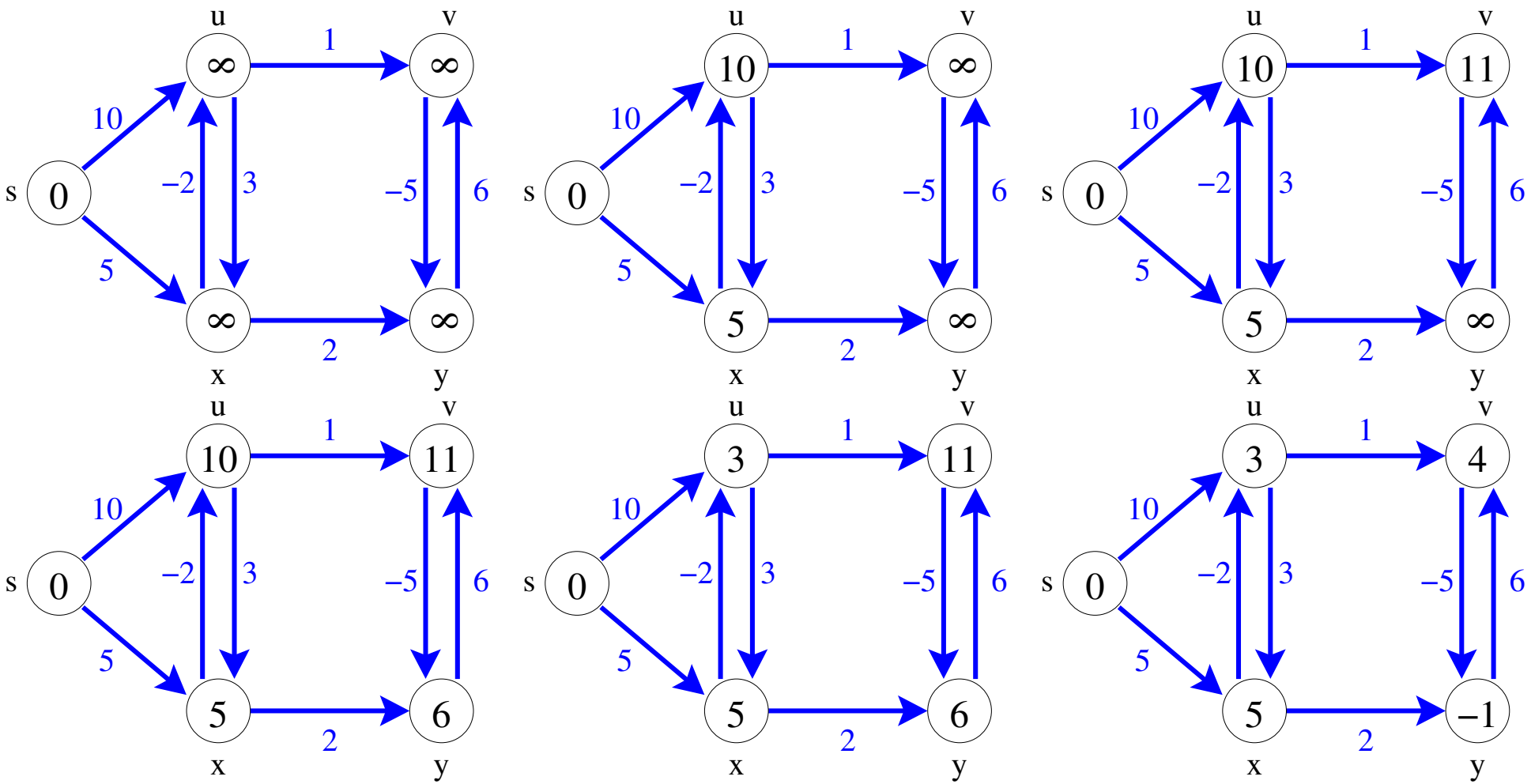
Kürzeste Wege mit negativen Kantengewichten (ohne negative Zyklen)

Bellmann-Ford-Algorithmus BF(G,s)

```
Fuer alle Knoten v setze D[v] = unendlich
D[s] = 0
Von i = 1 bis n-1
   Fuer alle Kanten (u,v)
      Falls D[u] + g(u,v) < D[v] dann
         D[v] = D[u] + g(u,v)
```

Enthält ein Graph Zyklen mit negativem Gewicht, gibt es keinen kürzesten Weg.
Der Bellmann-Ford-Algorithmus kann leicht modifiziert werden, um das Vorhandensein von Zyklen mit negativem Gewicht zu erkennen.
D[v] gibt den Abstand jeden Knotens zu s an.

Bellmann-Ford: Beispiel



(Kanten werden hier in lexikographischer Ordnung durchlaufen), also $(s,u)(s,x)(u,v)(u,x)(v,y)$

Zentralität

Gegeben ein Graph mit $V = (v_1, \ldots, v_n)$ Knoten. Wie können wir verschiedene Knoten nach ihrer „Zentralität“ unterscheiden?

Bisher haben wir kennengelernt

1. Grad: Zentrale Knoten haben mehr Nachbarn als weniger zentrale.
2. Exzentrizität: Derjenige Knoten mit dem geringsten Abstand zu dem am weitesten entfernten Knoten
3. Distanzsumme: Der Knoten, für den die Summe der Distanzen minimal ist, hat minimalen Durchschnittsabstand zu allen Knoten

Jetzt ausführlicher **ShortestPath Betweenness** und **PageRank**

Shortest Path Betweenness Zentralität

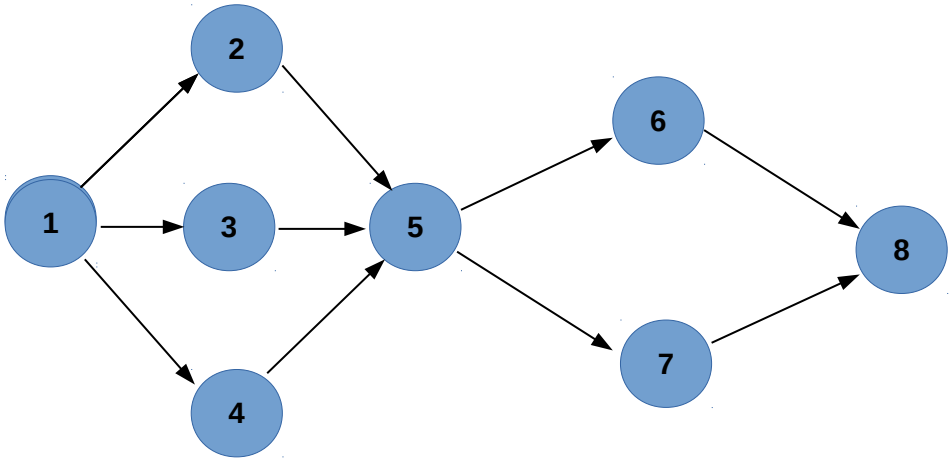
Idee: Wie stark ist ein Knoten an der Übertragung von Informationen im Netzwerk beteiligt?
Für einen Knoten $v \in V$ wird angenommen, dass er umso zentraler ist, je mehr er an möglichen Kommunikationswegen zwischen Paaren von Knoten beteiligt ist (unter der Annahme, dass als Kommunikationswege jeweils nur kürzeste Wege gewählt werden).

[FREEMAN, L.C.: A set of measures of centrality based on betweenness. Sociometry, 40(1):35–41, 1977.]

Für jedes Paar $s, t \in V$ von Knoten wird der Einfluss von v für die Kommunikation zwischen s und t gemessen als der Anteil aller kürzesten st-Wege, die v berühren. Die Kürzeste-Wege-Zentralität ist die Summe des Einflusses von v auf die Kommunikation zwischen allen Paaren von Knoten.

Shortest Path Betweenness Beispiel

Sei σ_{st} die Zahl der kürzesten Wege zwischen s und t, $\sigma_{st}(v)$ die Zahl der kürzesten Wege zwischen s und t, auf denen v als Zwischenknoten liegt.



Im Beispiel liegt Knoten 5 auf allen kürzesten Wegen von 1 nach 8, insgesamt 6.

Shortest Path Betweenness Definition

Ein Knoten ist **zentral**, wenn er auf einem hohen Anteil kürzester Wege liegt.

$B(v) := \sum_{s,t \in V, s \neq t, s,t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$

Ein hoher B-Wert bedeutet, dass ein Knoten auf einem hohen Anteil von kürzesten Pfaden liegt.
Im Beispiel: B(7) = 2,5 (da (1,8), (2,8), (3,8), (4,8) und (5,8) jeweils 0,5)

Ein Knoten v liegt genau dann auf dem kürzesten Weg zwischen s und t, wenn d(s,v)+d(v,t)=d(s,t) gilt.

Shortest Path Betweenness Algorithmus

Gegeben ein Graph $G = (V, E)$,

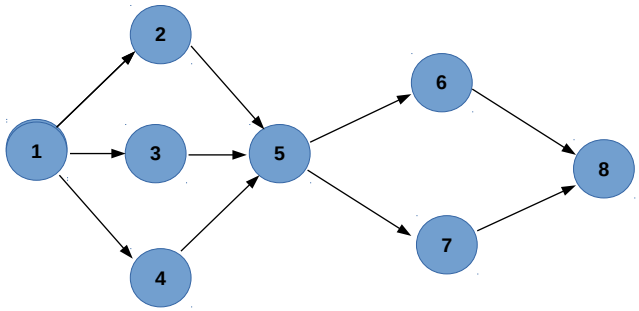
- Berechne Länge und Zahl der kürzesten Wege zwischen allen zulässigen Knotenpaaren mit Breitensuche (BFS)
- Berechne B-Werte für alle Knoten

Komplexitätsbetrachtung für jeden Knoten $s \in V$ lassen sich alle Distanzen d(s, t) im Graphen sowie die Anzahlen σ_{st} der kürzesten Wege zwischen allen Paaren von Knoten mit Hilfe der Breitensuche in O(nm) berechnen. Die einzelnen Summenglieder $\frac{\sigma_{st}(v)}{\sigma_{st}}$ lassen sich jeweils in O(1) berechnen. Durch einfaches Aufsummieren erhält man anschließend in $O(n^2)$ die Shortest Path Betweenness für einen Knoten v bzw. in $O(n^3)$ für alle Knoten,

Verbesserung

Beobachtung: Zwischen den Knoten bestehen **Paarabhängigkeiten**. Bestimmte Knoten müssen für einen Pfad von s nach t besucht werden. Wir nutzen d(s,v)+d(v,t)=d(s,t) und definieren die Paarabhängigkeit der kürzesten Wege von s nach t über v mit $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$

Die Betweenness-Zentralität können wir nunmehr bestimmen als die Summe der Paarabhängigkeiten
$B(v) := \sum_{s,t \in V, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} = \sum_{s,t \in V, s \neq t} \delta_{st}(v)$



Paarabhängigkeiten für Knoten 5: 1-6, 1-7, 1-8, 2-6, 2-7, 2-8, 3-6, 3-7, 3-8, 4-6, 4-7, 4-8

Verbesserung - Brandes (2001)

Wir definieren die **Vorgängermenge** von v bzgl. einer Quelle s $P_s(v)$ als die Menge der Nachbarn u von v, die auf einem kürzesten Weg von s nach v liegen. (Im Beispiel s $= 1$, v $= 8$, $P_1(8) = \{6, 7\}$).

Idee (verkürzt):

- Berechne mit Breitensuche und Dijkstra für jeden Startknoten $s \in V$ die Zahl und Länge der kürzesten Wege zu allen anderen Knoten und lege das Ergebnis in Form eines *Kürzeste-Wege-Baums* ab.
- Starte an den Blättern des Baumes und berechne für jedes $s \in V$ und alle anderen $v \in V$ die Abhängigkeiten $\delta_{s*}(v)$).
- Akkumuliere den Abhängigkeitswert des Startknotens s zu jedem einzelnen Knoten v im Zentralitätswert von v

Verbesserung - Komplexitätsbetrachtung

Zunächst werden nacheinander für jedes $s \in V$ die Paarabhängigkeiten $\delta_{s*}(v)$ berechnet. Mittels Breitensuche können diese in Zeit O(m) (ungewichtet) berechnet werden.

Da die Knoten in abnehmender Distanz zu s durchlaufen werden, wurden alle Knoten, die w als Vorgänger haben, schon vorher behandelt und müssen nicht noch mal neu berechnet werden.

Insgesamt sind die Berechnungen für n Knoten durchzuführen. Die Verbesserung ermöglicht es also, die Shortest Path Betweenness Centrality für alle Knoten mit einem Aufwand von O(nm) zu berechnen.

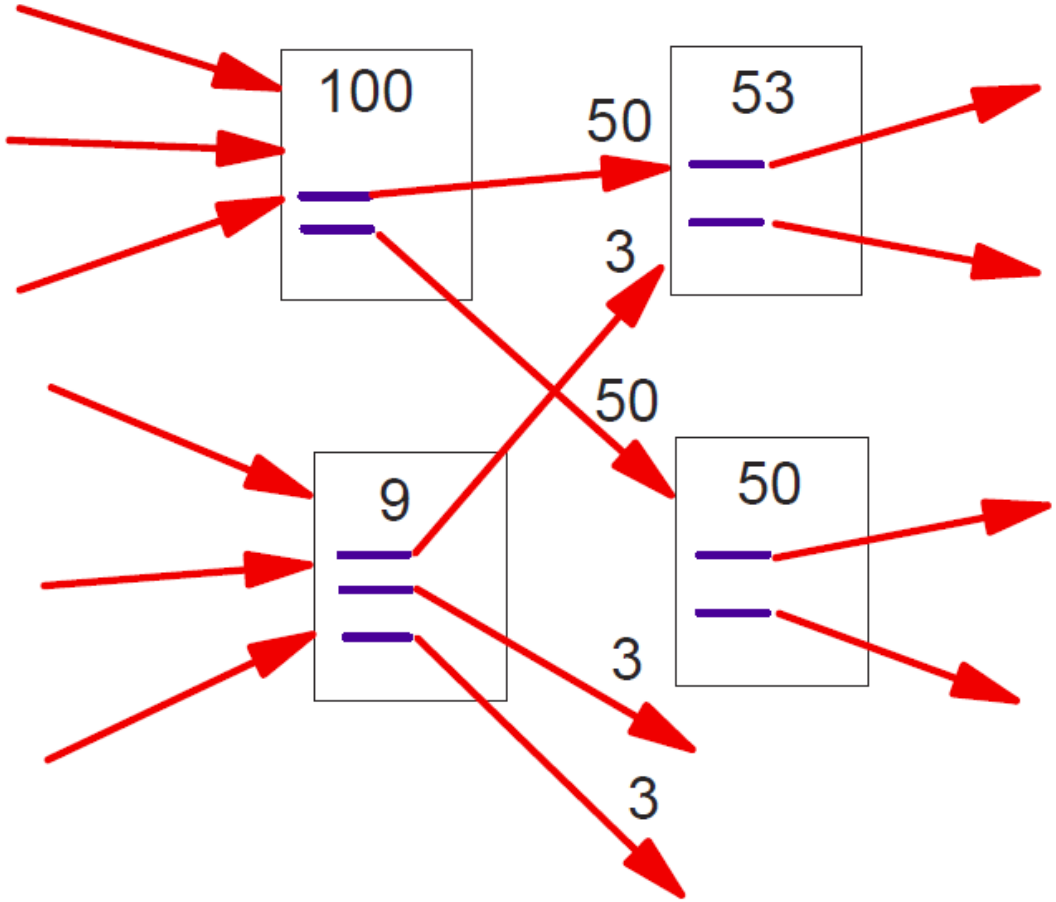
PageRank

In einem gerichteten Graphen G = (V,E) sind die Knoten miteinander *verlinkt*. Wir nutzen diese Linkstruktur, um die Zentralität von Knoten in dem Graphen zu berechnen.

Idee:

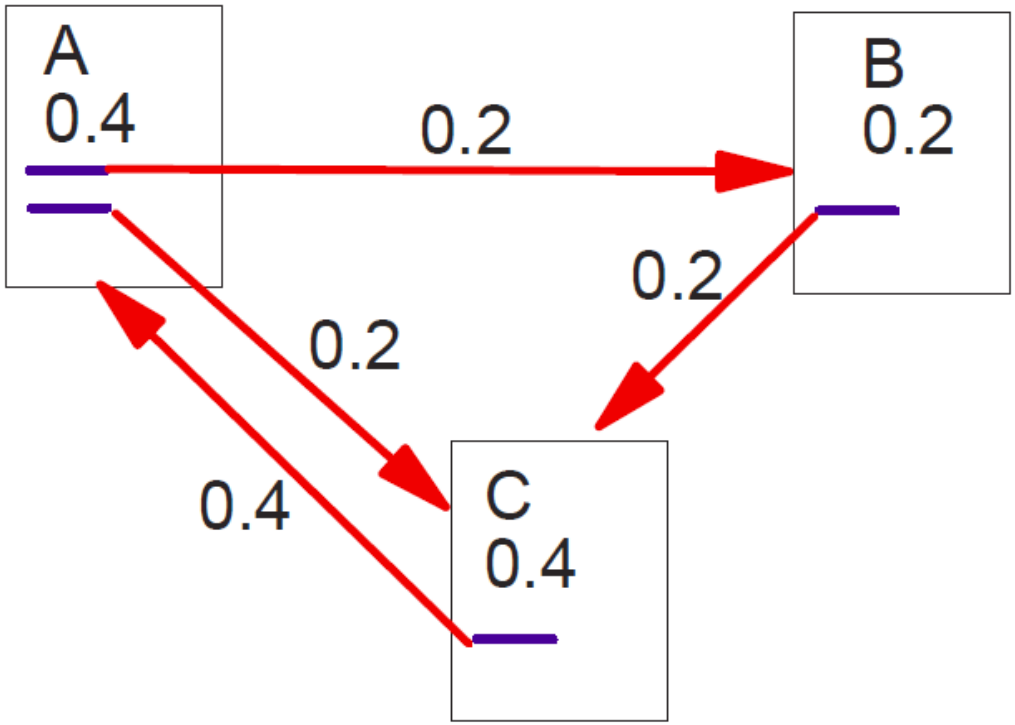
- Jeder Knoten hat einen PageRank und verweist auf eine bestimmte Anzahl anderer Knoten.
- Jeder der verlinkten Knoten hat selbst wiederum einen Pagerank.
- Die Links haben ein Gewicht in Abhängigkeit von dem PageRank des Knotens: Je höher der PageRank eines Knotens ist, desto höher ist das Gewicht seiner ausgehenden Kanten.
- Der PageRank eines Knotens verteilt sich gleichmäßig auf alle ausgehenden Kanten.

PageRank Beispiel: Verlinkung



Quelle: Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd, The PageRank Citation Ranking: Bringing Order to the Web", Technical Report, January 29th, 1998. Stanford InfoLab.

PageRank Beispiel: Verteilung von Gewichten



Quelle: Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd, The PageRank Citation Ranking: Bringing Order to the Web", Technical Report, January 29th, 1998. Stanford InfoLab.

PageRank: Vereinfachte Berechnung

Es sei G = (V, E) ein gerichteter und gewichteter Graph von verlinkten Webseiten mit

- $u, v \in V$
- F_v der Menge der Seiten auf die v zeigt
- B_u der Menge der Seiten, die auf u zeigen
- c ein Normalisierungsfaktor (bezogen auf die Anzahl der betrachteten Webseiten)
- $N_v = |F_v|$

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

PageRank: Kommentar

- Pagerank ist gleichmäßig auf die vorwärts verlinken Seiten verteilt
- die Berechnung erfolgt rekursiv
- Berechnung des Pagerank kann bei beliebigen Knoten begonnen werden, bis der Wert konvergiert

Aber:

- Wenn eine Seite auf zwei Seiten zeigt, die nur auf sich gegenseitig verlinken, entsteht eine Senke (weil nicht auf weitere Seiten verlinkt wird)

Deshalb angepasste Formel mit einem *Dämpfungsfaktor d* entsprechend dem *Random-Surfer-Modell*

$$R(u) = (1 - d) + d \sum_{v \in B_u} \frac{R(v)}{N_v}$$

PageRank: Dämpfungsfaktor d

- Dem Random-Surfer-Modell liegt die Annahme zugrunde, dass ein Surfer, der sich durchs Netz klickt, nicht in der Schleife verharrt, sondern auf eine andere Seite geht.
- Der Dämpfungsfaktor d modelliert das Nutzerverhalten, mit welcher Wahrscheinlichkeit ein Nutzer von einer Seite zu anderen Seiten wechselt. Dieser Wert ist empirisch zu bestimmen, in Brin und Page: *The Anatomy of a Large-Scale Hypertextual Web Search Engine* (1998) wird er mit 0,85 angegeben.
- Der Ausdruck 1 - d repräsentiert die Wahrscheinlichkeit, bei einem Random Walk im Webgraphen ausgehend von einer beliebigen Seite auf dem Knoten u zu landen
- Der Ausdruck $d \sum_{v \in B_u} \frac{R(v)}{N_v}$ repräsentiert die Wahrscheinlichkeit, ausgehend vom Knoten u, beim Knoten v zu landen. (d gibt also den Anteil von Page-Rank an, den eine Seite immer zu den von ihr verlinkten Seiten abgibt).

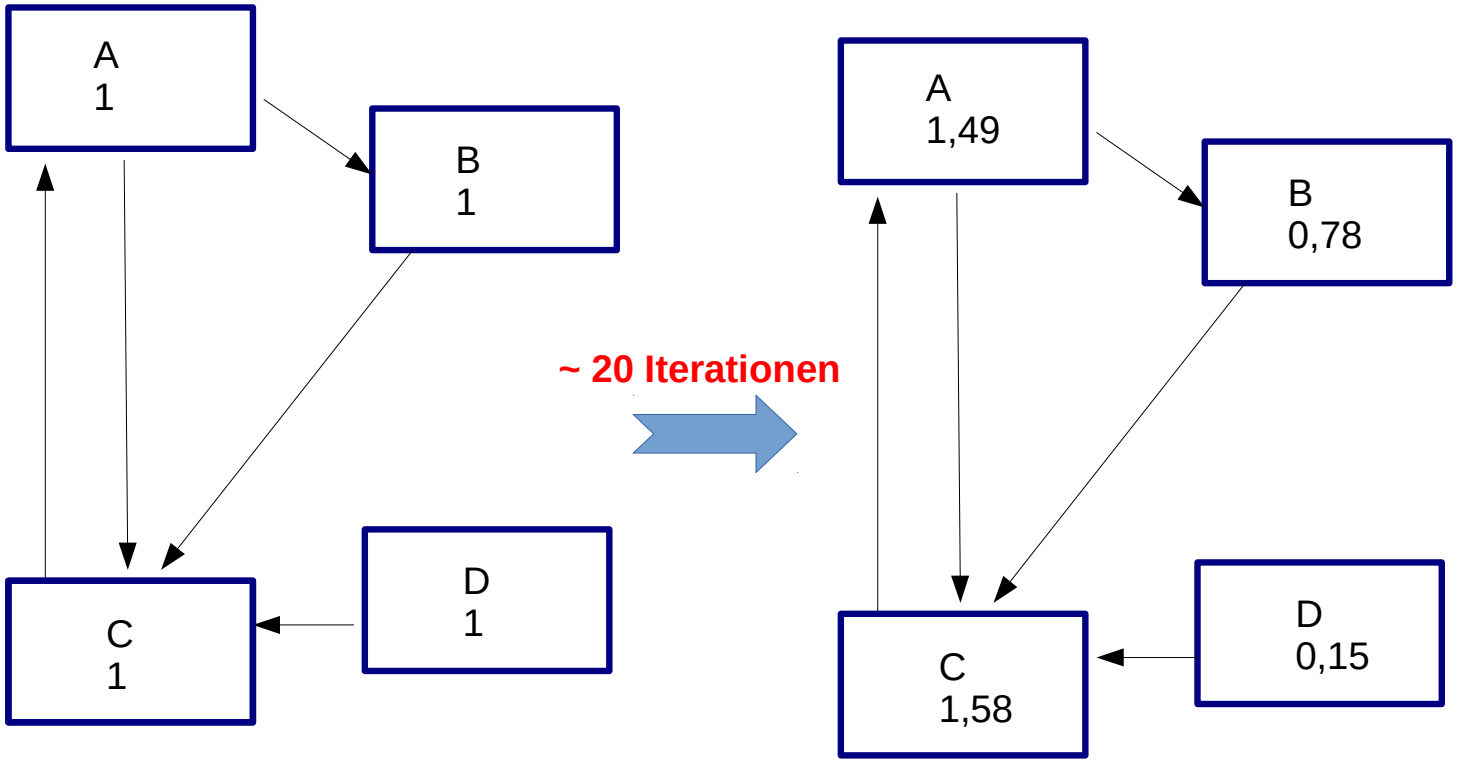
PageRank: Berechnung

Jede Webseite weist jeder von ihr verlinkten Seite einen bestimmten Anteil ihres eigenen Page-Ranks zu. Sei $|S_i|$ diese Anzahl der ausgehenden Links für eine Seite S_i.
Der Page-Rank einer Seite ist dann durch folgendes Gleichungssystem definiert und kann iterativ berechnet werden (für n von 1 bis i):
$R(S_i) := (1\text{-}d) + d\ (\ R(S_1)\ /\ |S_1| + \ldots + R(S_n)\ /\ |S_n|))$ (mit $|S_i| > 0$)

Verfahren:

- initialisiere jede Seite mit dem initialen Wert (1 - d)
- wähle eine beliebige Seite
- iteriere bis $|R(S_i)|_{j+1} - |R(S_i)|_j$ minimal
  - berechne (mit Breitensuche) alle Seiten, auf die S_i verlinkt
  - berechne für jede verlinkte Seite ihren Page-Rank R

PageRank: Beispiel



PageRank: Kurze Historie

Versionen und Weiterentwicklungen:

- 1998: Veröffentlichung des PageRank-Algorithmus von Brin und Page
- 2010: Rational Surfer Modell
- 2013: neuer Algorithmus - Hummingbird