ADS: Algorithmen und Datenstrukturen 2

Teil 5

Prof. Dr. Gerhard Heyer

Institut für Informatik
Abteilung Automatische Sprachverarbeitung
Universität Leipzig

8. Mai 2019

[Letzte Aktualisierung: 15/05/2019, 13:10]

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 - Vorschlag von Floyd

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

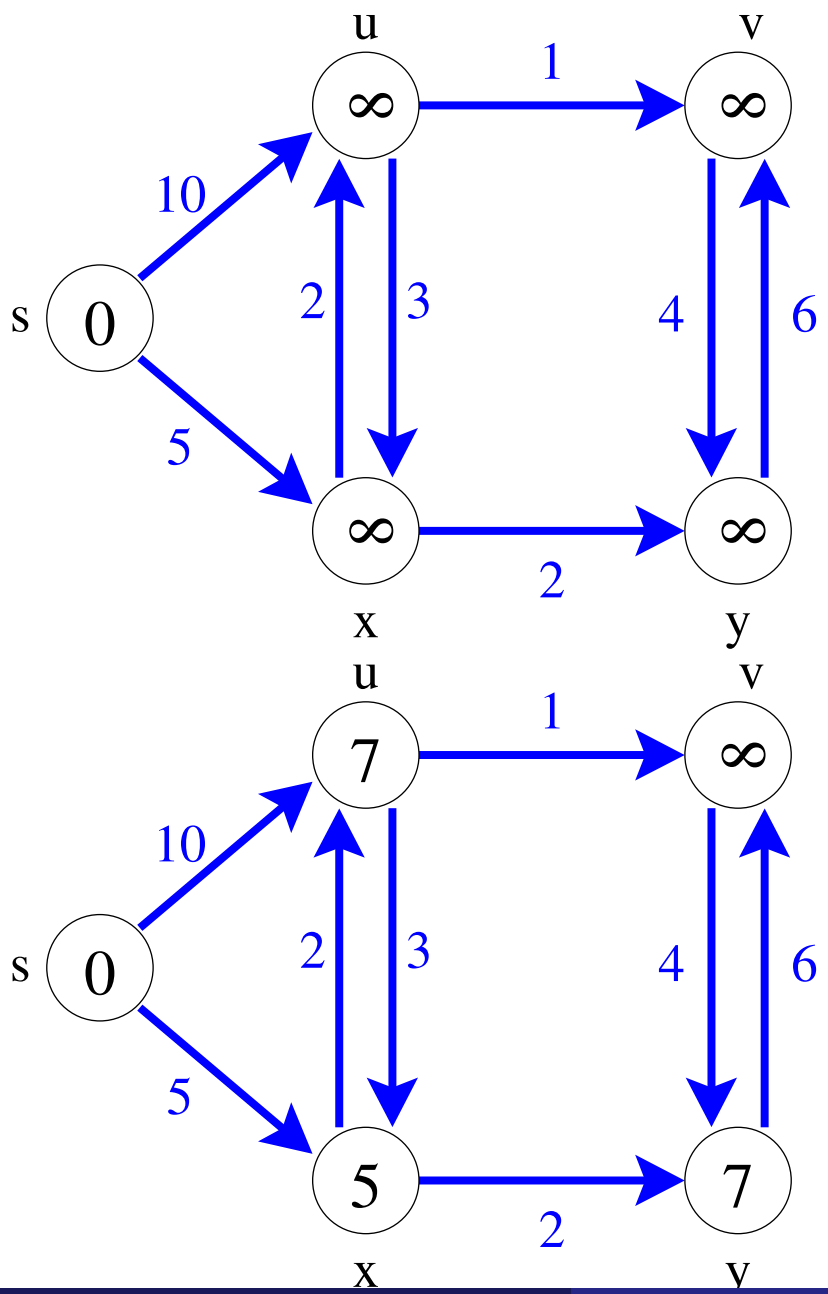
Bestimmung der von einem Knoten ausgehenden kürzesten Wege

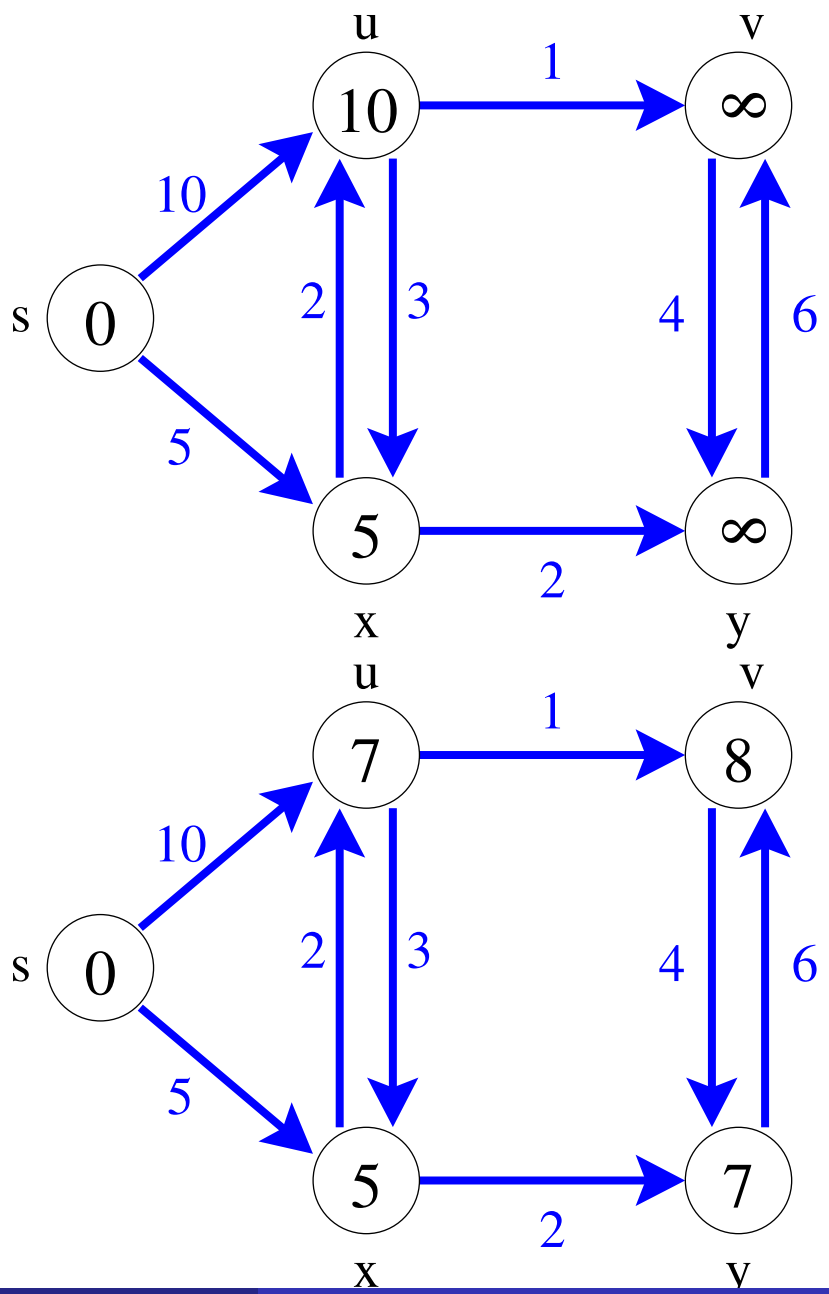
- gegeben: kanten-bewerteter Graph $G = (V, E, g)$ mit $g : E \to \mathbb{R}_0^+$ (Kantengewichte)
- Startknoten s; zu jedem Knoten u wird die Distanz zu Startknoten s in $D[u]$ geführt
- Q sei Prioritäts-Warteschlange (sortierte Liste);
  Priorität = Distanzwert
- Funktion $succ(u)$ liefert die Menge 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)

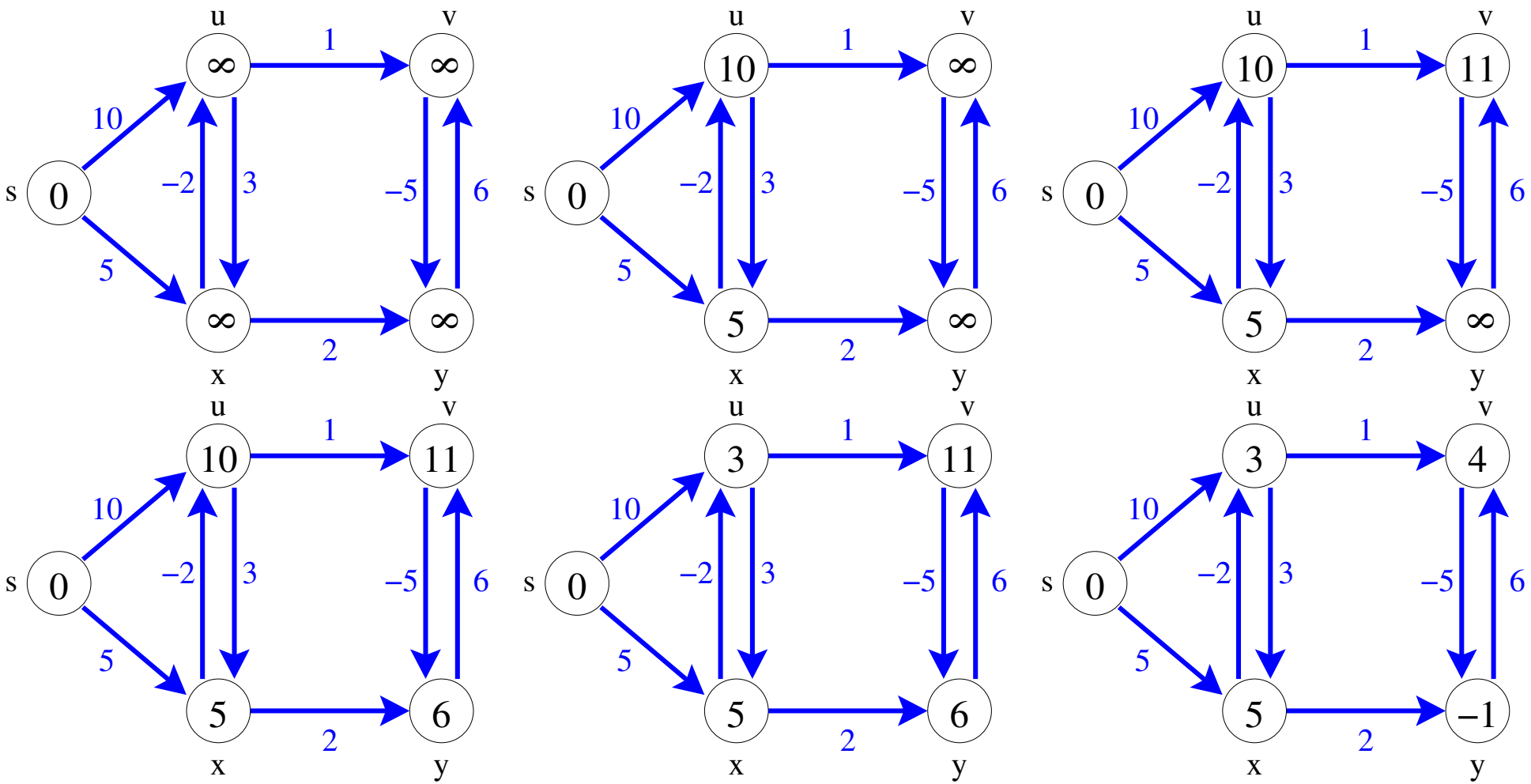
Kürzeste Wege mit negativen Kantengewichten (ohne negative Zyklen)

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

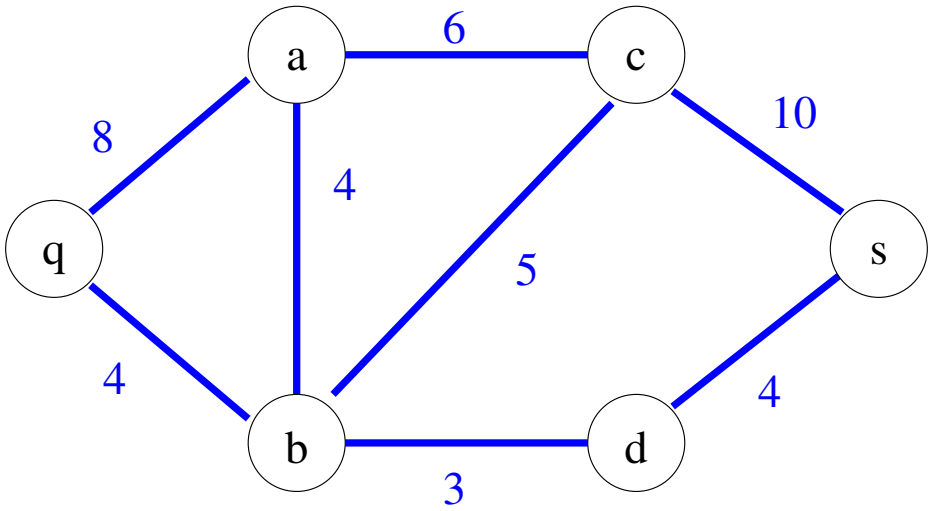


(Kanten werden hier in lexikographischer Ordnung durchlaufen), also $(s,u)(s,x)(u,v)(u,x)(v,y)$

Flüsse in Netzwerken I

Anwendungsprobleme:

- Wieviel Autos können durch ein Straßennetz fahren?
- Wieviel Abwasser fasst ein Kanalnetz?
- Wieviel Strom kann durch ein Leitungsnetz fließen?

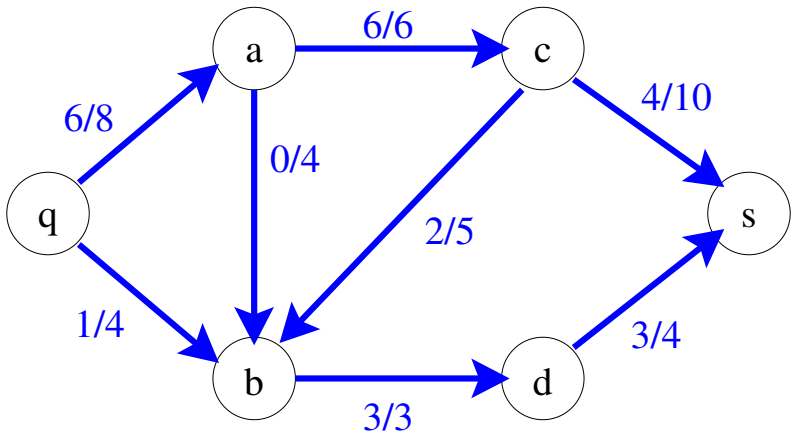


Kantenwert: Kapazität $c(e)$

Flüsse in Netzwerken II

- Ein (Fluss-) Netzwerk ist ein gerichteter Graph $G = (V, E, c)$ mit ausgezeichneten Knoten q (Quelle) und s (Senke), sowie einer Kapazitätsfunktion $c :\rightarrow \mathbb{R}^+$.
- Ein Fluss für das Netzwerk ist eine Funktion $f : E \rightarrow \mathbb{R}$, so dass gilt:
  - Kapazitätsbeschränkung: $\forall e \in E : 0 \leq f(e) \leq c(e)$.
  - Symmetriebedingung: $f(u, v) = -f(v, u)$
  - Flusserhaltung/Kirchhoffsches Gesetz: $\forall v \in V \setminus \{q, s\} : \sum_{(v',v)\in E} f(v', v) = \sum_{(v,v')\in E} f(v, v')$
  - Der Wert von f, $w(f)$, ist die Summe der Flusswerte der die Quelle q verlassenden Kanten: $\sum_{(q,v)\in E} f(q, v)$

Flüsse in Netzwerken III



Kantenbeschriftung: Fluss $f(e)$ / Kapazität $c(e)$

Gesucht: Fluss mit maximalem Wert

- begrenzt durch Summe der aus q wegführenden bzw. in s eingehenden Kapazitäten

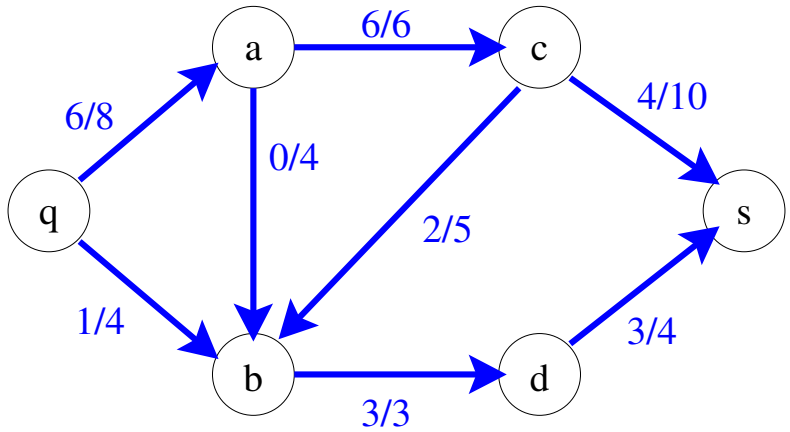
Restgraph

Sei f ein zulässiger Fluss für $G = (V, E, c)$. Sei
$E' = \{(v, w) | (v, w) \in E \vee (w, v) \in E\}$

- Wir definieren die Restkapazität einer Kante $e = (v, w) \in E'$ als

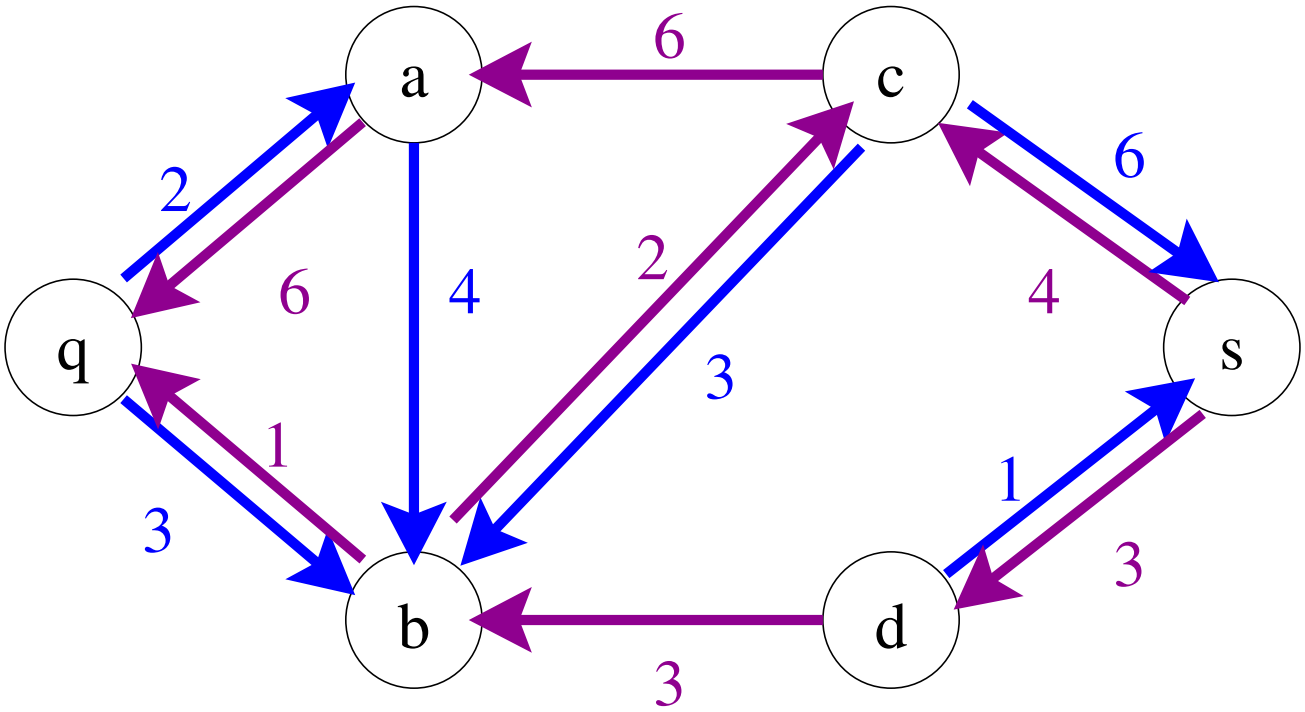
$$rest(e) := \begin{cases} c(e) - f(e) & \text{falls } e \in E \\ |f(\bar{e})| & \text{falls } \bar{e} \in E \end{cases}$$

  wobei $\bar{e} = (w, v)$ die Rückwärtskante zu e bezeichne.
- Der Restgraph G_f von f (bzgl. G) besteht aus den Kanten $e \in E'$, für die $rest(e) > 0$



$f(e)/c(e)$

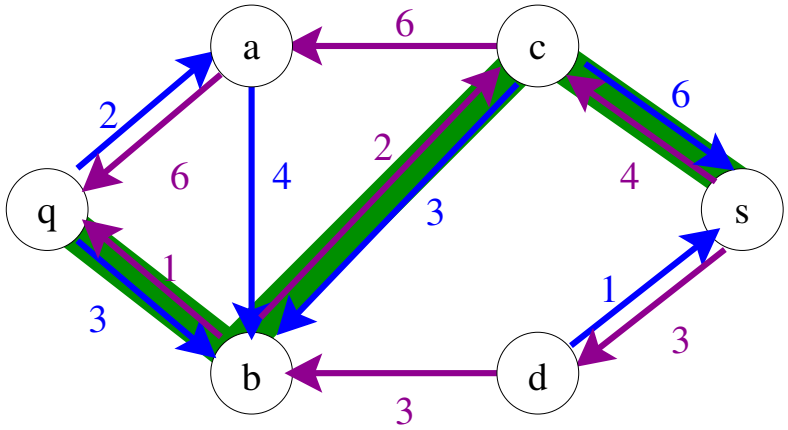
Restgraph



$rest(e)$

Erweiterungspfad

- Jeder gerichtete Pfad p von q nach s im Restgraphen heißt Erweiterungspfad.

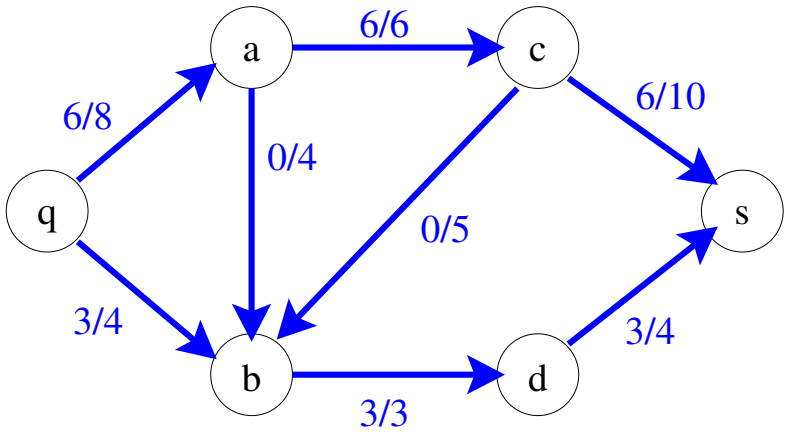


Ein *Erweiterungspfad* ist ein einfacher (zyklenfreier) Pfad p in G_f von q nach s.
Die *Restkapazität* von p ist $c_f(p) = min\{(c_f(u,v)|(u,v) \text{ liegt auf } p\}$

Erweiterungspfad

Beobachtung:
Sei G ein Netzwerk, c eine Kapazität und f ein Fluss. Sei G_f das zugehörige Restnetzwerk und c_f die zugehörige Restkapazität. Sei ferner g ein zulässiger Fluss auf G_f. Dann gilt: $(f + g)$ ist ein zulässiger Fluss auf G mit $|(f + g)| = |f| + |g|$

- Optimierung des Flusses für G:
  - entlang eines zunehmenden Weges p
  - Verbesserung um minimales $rest(e)$ aller e auf dem Weg p



Ford-Fulkerson-Algorithmus

Optimierung eines bestehenden Flusses f in einem Netzwerk G mit Kapazität c:

- bestimme Restnetzwerk G_f
- suche Erweiterungspfad
- bestimme $c_f(p)$
- erhöhe den Fluss um Minimum der verfügbaren Restkapazität der einzelnen Kanten des Erweiterungspfades

```
Solange es einen zunehmenden Weg p in Restgraph G gibt
   r = min {rest(e) | e liegt auf p}
   Fuer alle e = (v,w) auf p tue
      Falls e in E dann
         f(e) = f(e) + r
      Sonst f(w,v) = f(w,v) - r
```

Ford-Fulkerson-Algorithmus — Laufzeit

- $O(|E|)$ für Konstruktion des Restgraphen
- $O(|E|)$ für Konstruktion eines Erweiterungspfades: Graphtraversierung und Bestimmung des Flusses entlang des Wegs
- in jedem Schritt verbessert sich der Fluss um $r = \min\{rest(e) | e \text{ liegt auf } p\}$

Für *ganzzahlige* Kapazitäten: $O(E) \times \max f$ (jeder Durchlauf erhöht den Fluss um mindestens eins, also gibt es bis zu $|f^*|$ Durchläufe
(f* maximaler Fluss)

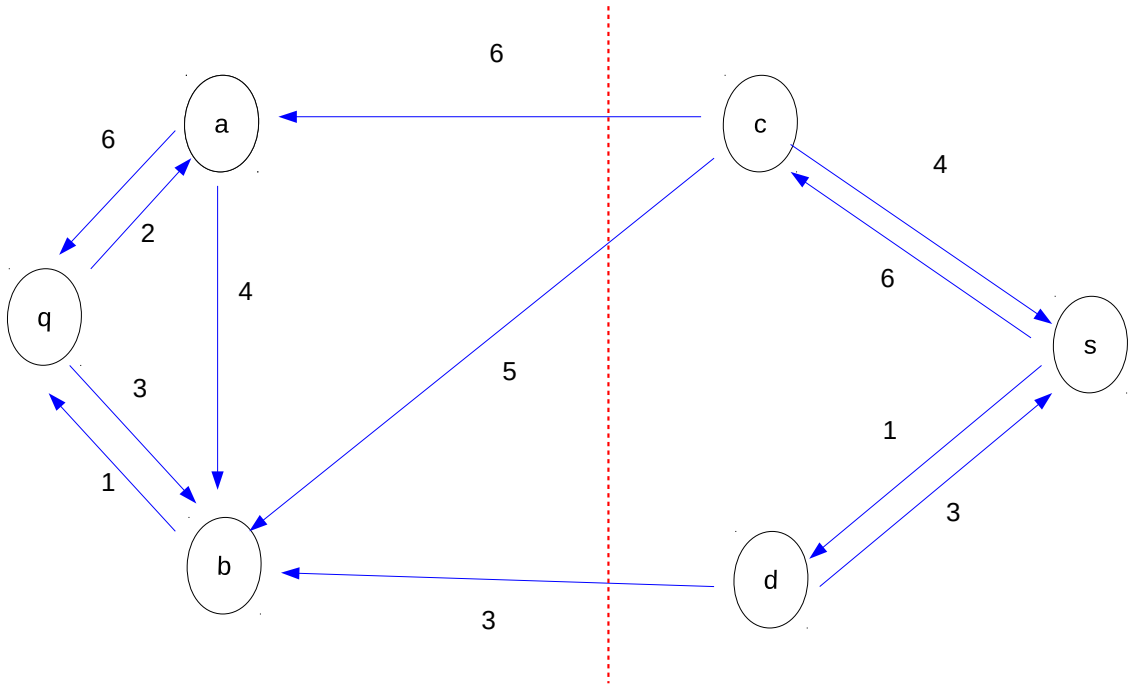
$\Rightarrow$ Für *allgemeine* Kapazitäten: Konvergenz nicht garantiert!

Min-Cut

Wenn ein Erweiterungspfad gefunden werden kann, bringt der neue Fluss $(f + g)$ eine Verbesserung.

Aber kann immer ein Erweiterungspfad gefunden werden, wenn der bisherige Fluss f nicht maximal ist?

Beobachtung: Im Graph mit optimalem Fluss existiert im Restgraph kein Weg von Quelle zu Senke!



Min-Cut-Max-Flow-Theorem

Es sei V ein Graph bestehend aus der Menge der Kanten $e = (u, v)$ mit $u \in A$ und $v \in B$. Wir definieren *Schnitt* (A, B) mit $A \cap B = \emptyset$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cup B = V$,
Die Kapazität von (A, B) is $c(A, B) = \sum_{e \in (A,B)} c(e)$.

Theorem (Min-Cut-Max-Flow-Theorem): Sei f ein zulässiger Fluss für G. Dann sind folgende Aussagen äquivalent:

1. f ist maximaler Fluss in G.
2. Der Restgraph von f enthält keinen zunehmenden Weg.
3. $w(f) = c(A, B)$ für einen Schnitt (A, B) von G.

Min-Cut-Max-Flow-Theorem – Beweis

(1) Sei (A, B) ein Schnitt mit $q \in A$ und $s \in B$. Flusserhaltung impliziert, dass $\sum_{e \in (A,B)} f(e) = w(f)$. Dies gilt für jeden Schnitt, der q und s trennt. Da $f(e) \leq c(e)$ erfüllt sein muss, gilt ausserdem $w(f) \leq c(A, B)$ für alle Schnitte die q und s trennen.

(2) ((1) $\Rightarrow$ (2)) Sei f ein maximaler Fluss. Würde der zugehörige Restgraph G_f einen zunehmenden Weg enthalten, könnte der Fluss weiter erhöht werden. Also gibt es keinen zunehmenden Weg.

(3) ((2) $\Rightarrow$ (3)) Sei A die Menge der Knoten, die in G_f von q aus erreicht werden können, und sei $B = V \setminus A$. Also gilt $q \in A$ und insbesonders $s \in B$. Zudem ist (A, B) ein Schnitt, der q und s trennt.
Es gibt keinen Weg in G_f durch Kanten von $c(A, B)$. (Wenn $e = (u, v) \in (A, B)$ existieren würde, wäre ja nicht nur u von q erreichbar sondern auch v, was der Definition von B widersprechen würde.) Daher gilt für die Restkapaziät entlang jeder Kante von $e \in c(A, B)$, dass $rest(e) = c(e) - f(e) = 0$. Also ist $f(e) = c(e)$ für alle $e \in c(A, B)$.
Wegen (1) gilt $w(f) = \sum_{e \in (A,B)} f(e) = \sum_{e \in (A,B)} c(e) = c(A, B)$.

(4)((3) $\Rightarrow$ (1)) Für alle zulässigen Flüsse f und Schnitte (A, B), die q und s trennen, gilt nach (1) $w(f) \leq c(A, B)$. Wenn also $w(f) = c(A, B)$ ist, dann ist $w(f)$ notwendigerweise maximal.

Matching

Matching (Zuordnung) M für ungerichteten Graphen $G = (V, E)$ ist eine Teilmenge der Kanten, so dass jeder Knoten in V in höchstens einer Kante vorkommt

- $|M|$ = Größe der Zuordnung
- Perfektes Matching: kein Knoten bleibt "allein" (unmatched), d.h. jeder Knoten ist in einer Kante von M vertreten

Matching M ist maximal, wenn es kein Matching M' gibt mit $|M| < |M'|$

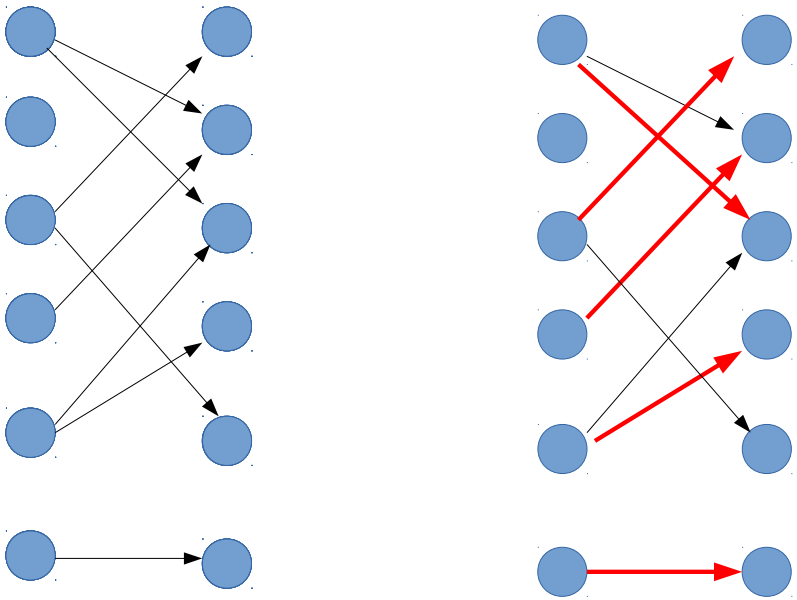
Ein bipartiter Graph ist ein Graph, dessen Knotenmenge V in zwei disjunkte Teilmengen V_1 und V_2 aufgeteilt ist, und dessen Kanten jeweils einen Knoten aus V_1 mit einem aus V_2 verbinden (also keine Kanten innerhalb von V_1 oder V_2).

Maximales Bipartites Matching

Beispiel (leicht verallgemeinerbar):

- Eine Gruppe von M Personen bewirbt sich auf N Jobangebote
- Jede Person hat eine Prioritätenliste von präferierten Jobs
- Jeder Arbeitgeber hat eine Proritätenliste von präferierten Kandidaten
- Wieviele Personen können maximal auf die verfügbaren Jobs verteilt werdeen?

Maximales Bipartites Matching - Beispiel



Verteilung von Bewerbern auf Jobangebote

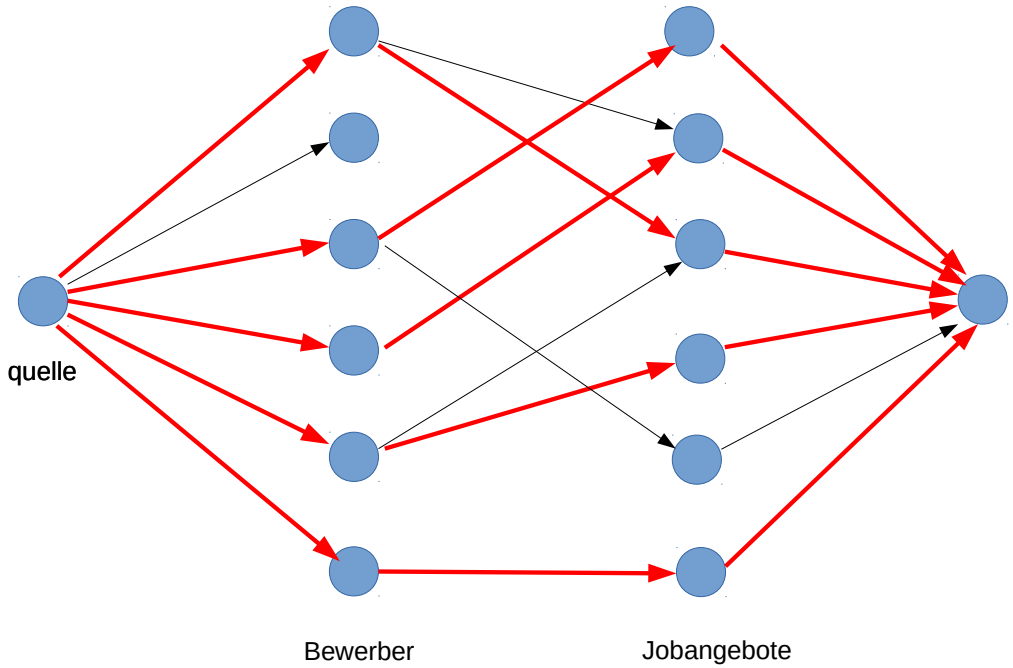
Bipartites Matching und maximaler Fluss

Maximales Matching kann auf maximalen Fluss zurückgeführt werden:

- Quelle und Senke hinzufügen.
- Kanten von V_1 nach V_2 richten.
- Jeder Knoten in V_1 erhält eingehende Kante von der Quelle.
- Jeder Knoten in V_2 erhält ausgehende Kante zur Senke.
- Alle Kanten erhalten Kapazität $c(e) = 1$

$\rightarrow$ Anwendung des Ford-Fulkerson-Algorithmus

Maximales Bipartites Matching - Beispiel



Verteilung von Bewerbern auf Jobangebote