ADS: Algorithmen und Datenstrukturen 2

Der Tragödie IV. Theyl

Peter F. Stadler & Konstantin Klemm

Bioinformatics Group, Dept. of Computer Science & Interdisciplinary Center for Bioinformatics, **University of Leipzig**

28. April 2010

Kürzeste Wege

- kantenmarkierter (gewichteter) Graph G = (V, E, g)
  - Weg/Pfad P der Länge n: $(v_0, v_1), (v_1, v_2), , \ldots, (v_{n-1}, v_n)$
  - 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
  - nichtnegative 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:
    es existiert kein Weg;
    es existiert Zyklus mit negativem Gewicht

Zur Erinnerung: Warshall's Transitive Hülle

```
A = adjacency matrix of G(V, E)
for i = 0; i < |V|; i++ do
    A[i][i] = "true"
end
for j = 0; j < |V|; j++ do
    for i = 0; i < |V|; i++ do
        if A[i][j] then
            for k = 0; k < |V|; k++ do
                if A[j][j] then
                    A[i][k] = "true"
                end
            end
        end
    end
end
```

Algorithm 1: *Warshall*(*A*)

Warshall's Algorithmus für Distanzen

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

if $((v_i, v_j) \in E)$ **then** $A[i][j] = g((v_i, v_j))$
else $A[i][j] = \infty$;
for $i = 0;\ i < |V|;\ i++$ **do** $A[i][i] = 0$;
for $j = 0;\ j < |V|;\ j++$ **do**
    for $i = 0;\ i < |V|;\ i++$ **do**
        for $k = 0;\ k < |V|;\ k++$ **do**
            if $(A[j][j] + A[j][k] < A[i][k])$ **then**
            $A[i][k] = A[j][j] + A[j][k]$
        end
    end
end

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

Dijkstra-Algorithmus I

Bestimmung der von einem Knoten ausgehenden kürzesten Wege

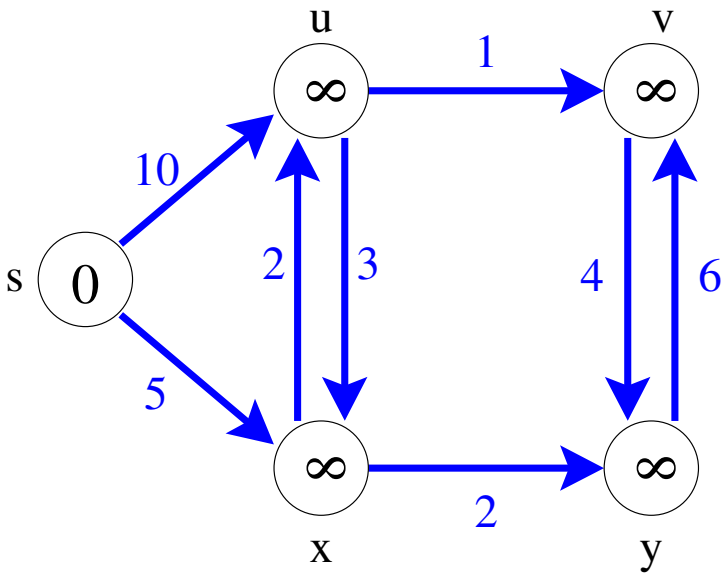
- gegeben: kanten-bewerteter Graph $G = (V, E, g)$ mit $g : E \rightarrow R+$ (Kantengewichte)
- Startknoten s; zu jedem Knoten u wird die Distanz zu Startknoten s in $D[u]$ geführt
- Q sei Prioritäts-Warteschlange (sortierte Liste); Priorität = Distanzwert
- Funktion $succ(u)$ liefert die Menge der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

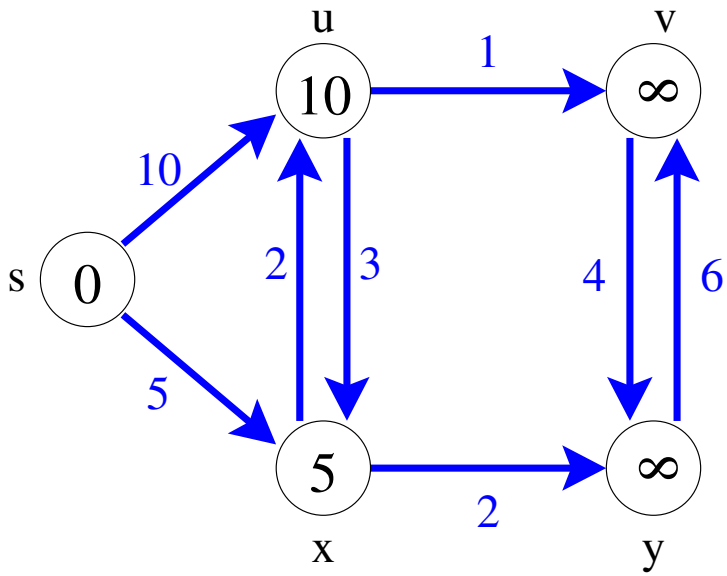
Dijkstra-Algorithmus II

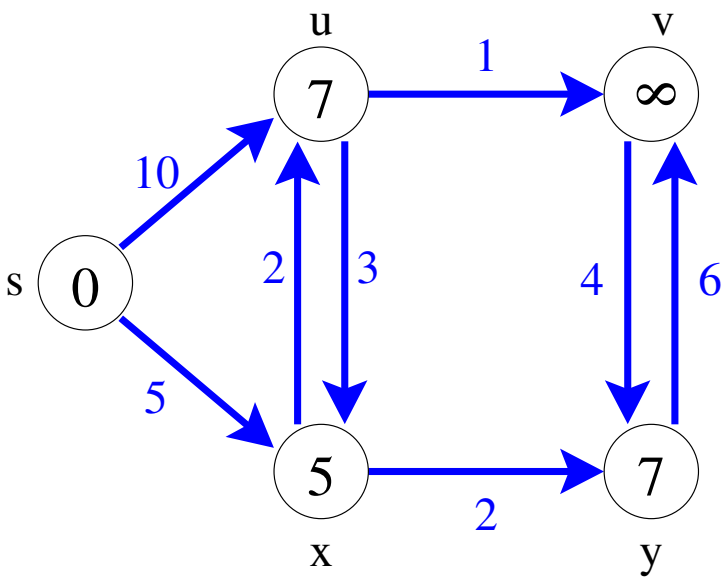
foreach *Knoten* $v \in V - s$ **do** $D[v] = \infty$;
$D[s] = 0$; PriorityQueue Q = V;
while *NOT isEmpty(Q)* **do**
    v = extractMinimum(Q);
    foreach *u in succ(v)* $\cap Q$ **do**
        if $D[v] + g((v,u)) < D[u]$ **then**
            $D[u] = D[v] + g((v,u))$;
            adjustiere Q an neuen Wert $D[u]$;
        end
    end
end

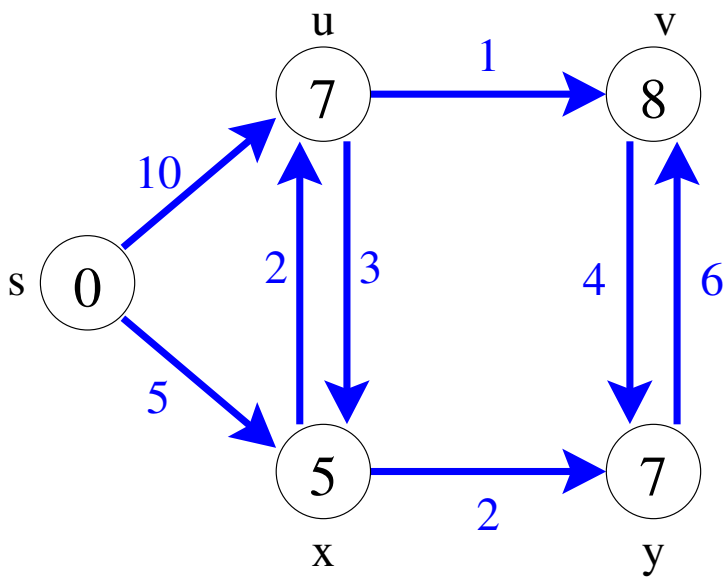
Algorithm 2: *Dijkstra*(G, s)

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 nichtnegative Kantengewichte (steigende Länge durch hinzugenommene Kanten)

Dijkstra-Algorithmus: Eigenschaften

Komplexität $\leq O(n^2)$

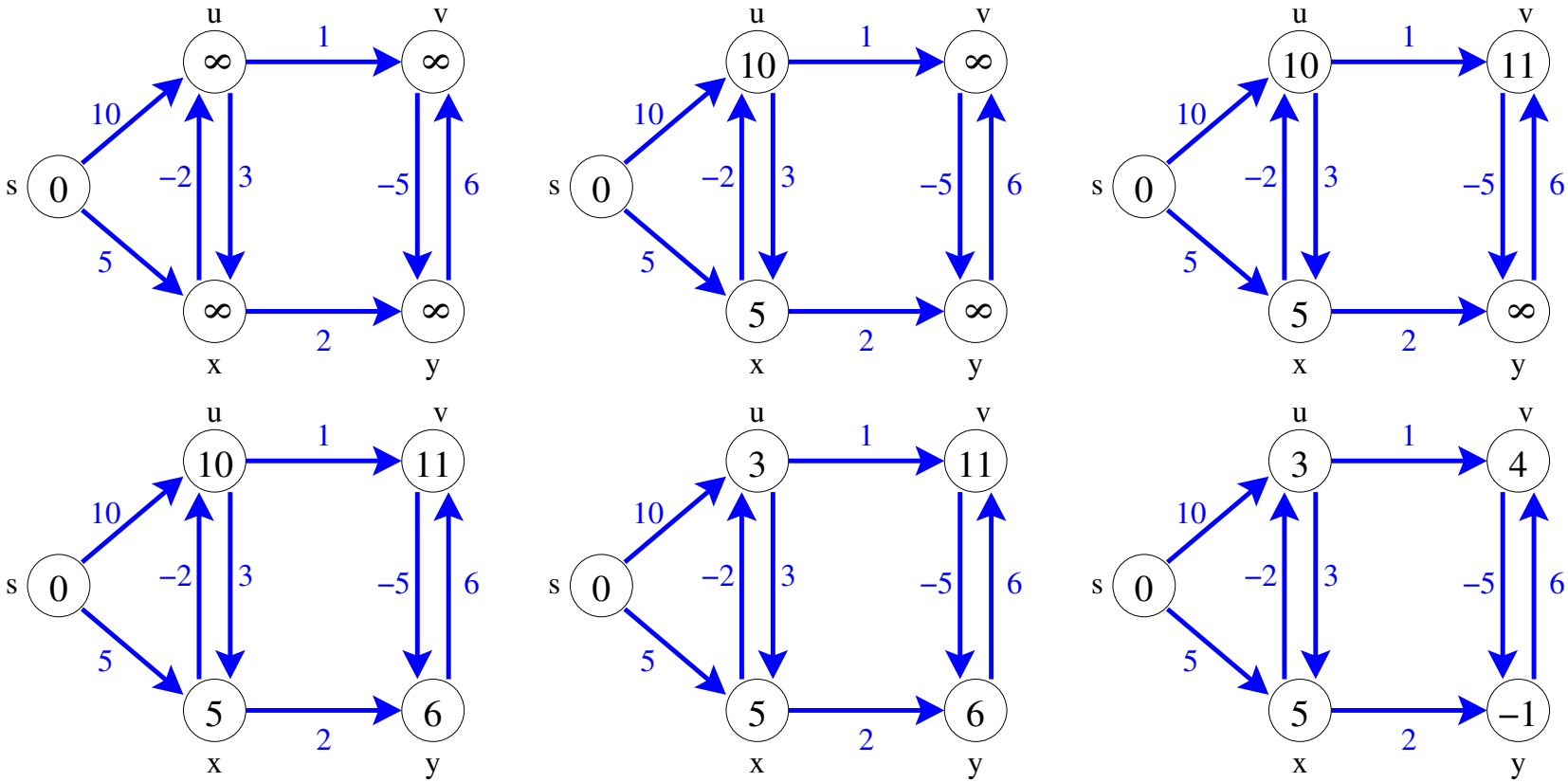
- n-maliges Durchlaufen der äußeren Schleife liefert Faktor $O(n)$
- innere Schleife: Auffinden des Minimums begrenzt durch $O(n)$, ebenso das Aufsuchen der Nachbarn von v
- Pfade bilden aufspannenden Baum (der die Wegstrecken von s aus gesehen minimiert)
- Bestimmung des kürzesten Weges zwischen u und v: Spezialfall für Dijkstra-Algorithmusmit Start-Knoten u (Beendigung sobald v aus Q entfernt wird)

Kürzeste Wege mit negativen Kantengewichten (ohne negative Zyklen

foreach *Knoten* $v \in V - s$ **do** $\{\mathrm{D[v]} = \infty\}$; $D[s] = 0$;
for *i=1 to* $|v| - 1$ **do**
    foreach *(u,v)*$\in E$ **do**
        if $D[u] + g((u, v)) < D[v]$ **then** $D[v] = D[u] + g((u, v))$
    end
end

Algorithm 3: Bellmann-Ford-Algorithmus BF(G,s)

Bellmann-Ford: Beispiel



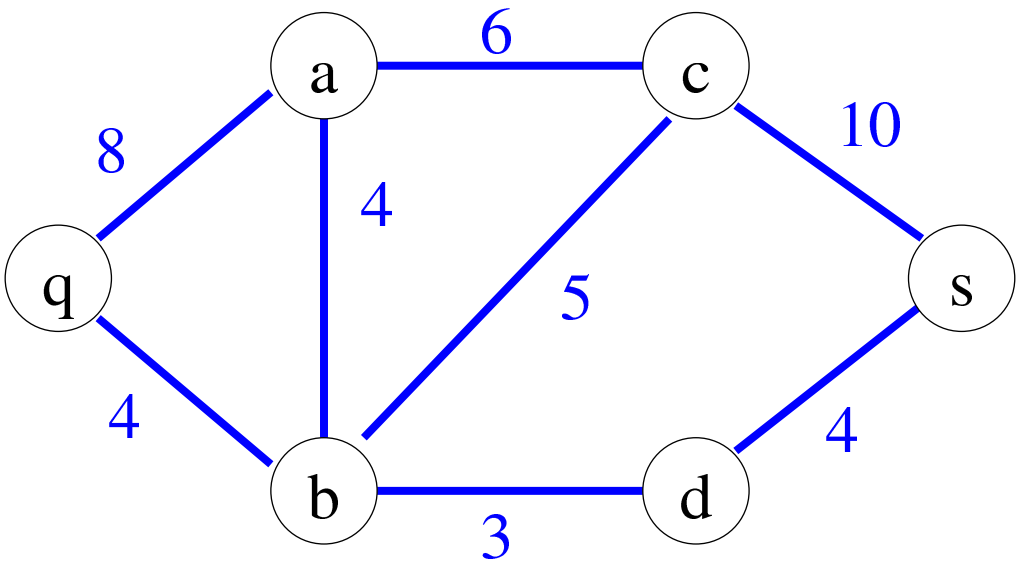
(Kanten werden hier in lexicographischer Ordnung durchlaufen), also

$(s, u)(s, x)(u, v)(u, x)(v, y)$

Flüsse in Netzwerken I

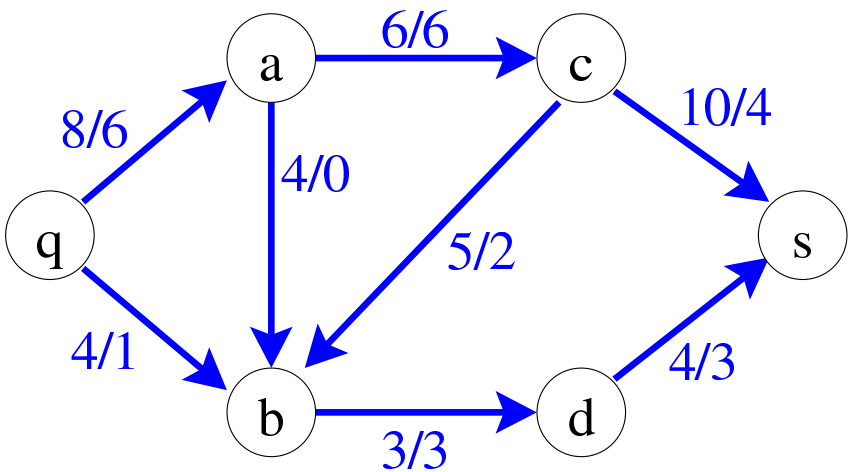
Anwendungsprobleme:

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



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 $c :\longrightarrow Z+$.
- Ein Fluß für das Netzwerk ist eine Funktion $f : E \longrightarrow Z$, so daß gilt:
  - Kapazitätsbeschränkung: $\forall e \in E : f(e) \leq c(e)$.
  - Flußerhaltung:
    $\forall v \in V \setminus \{q, s\} : \sum_{(v',v)\in E} f((v', v)) = \sum_{(v,v')\in E} f((v, v'))$
  - 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

Gesucht: Fluß mit maximalem Wert

- begrenzt durch Summe der aus q wegführenden bzw. in s eingehenden Kapazitäten
- jeder weitere "Schnitt" durch den Graphen, der q und s trennt, begrenzt max. Fluss

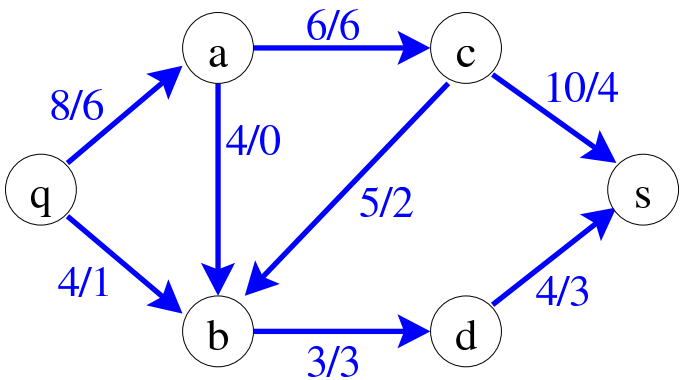
Schnitt (A, B) eines Fluß-Netzwerks ist eine Zerlegung von V in disjunkte Teilmengen A und B, so daß $q \in A$ und $s \in B$.

- Die Kapazität des Schnitts ist $c(A, B) = \sum_{u \in A, v \in B} c((u, v))$
- minimaler Schnitt (minimal cut): Schnitt mit kleinster Kapazität

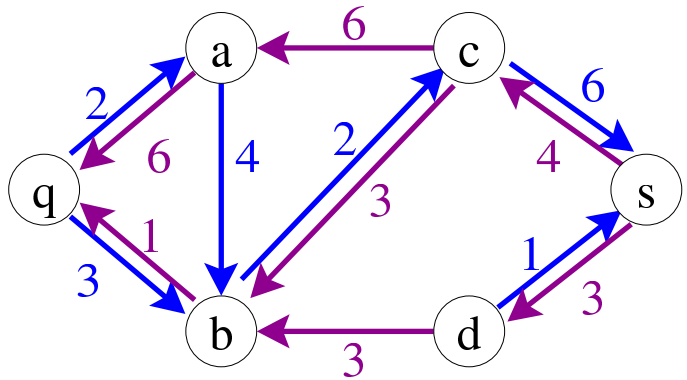
Restgraph

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

- Wir definieren die Restkapazität einer Kante (v, w) wie folgt:
  $rest((v, w)) = c((v, w)) - f((v, w))$ falls $(v, w) \in E$
  $rest((v, w)) = +f((w, v))$ falls $(w, v) \in E$
- Der Restgraph von f (bzgl. G) besteht aus den Kanten $e \in E'$, für die $rest(e) > 0$



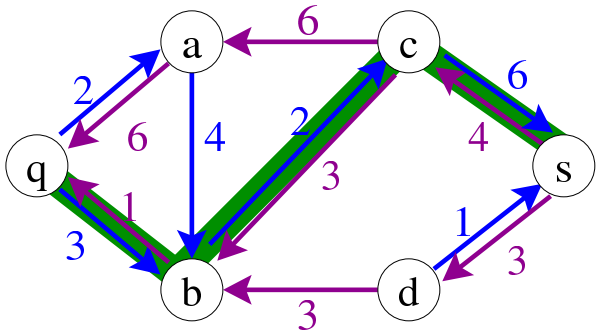
$c(e)/f(e)$



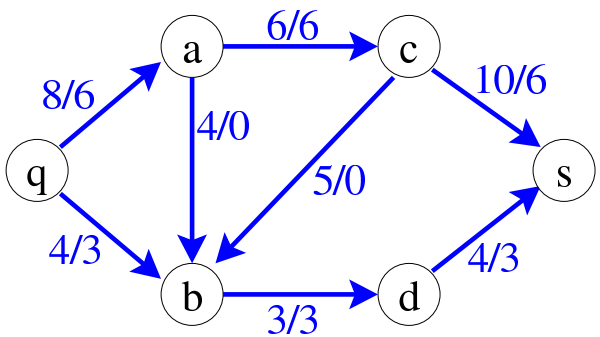
rest(e)

Zunehmender Weg

- Jeder gerichtete Pfad von q nach s im Restgraphen heißt zunehmender Weg.



- Optimierung des Flusses für G:
  - entlang des zunehmenden Weges q
  - um $min(rest(e))$ mit e liegt auf q



Min-Cut-Max-Flow-Theorem

Theorem (Min-Cut-Max-Flow-Theorem): Sei f zulässiger Fluß für G. Folgende Aussagen sind äquivalent:

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

Ford-Fulkerson-Algorithmus

Ford-Fulkerson-Algorithmus:

- füge solange zunehmende Wege zum Gesamtfluß hinzu wie möglich
- Kapazität erhöht sich jeweils um Minimum der verfügbaren Restkapazität der einzelnen Kanten des zunehmenden Weges

foreach $e \in E$ **do** f(e) = 0;
while *es gibt zunehmenden Weg p im Restgraphen von G* **do**
    $r = min(rest(e)|e$ liegt auf $p)$;
    foreach $e = (v, w)$ *auf p* **do**
        if $e \in E$ **then** $f(e) = f(e) + r$;
        else $f((w, v)) = f((w, v))–r$;
    end
end

Maximales Bipartites Matching

Beispiel:

- Eine Gruppe von Erwachsenen und eine Gruppe von Kindern besuchen Disneyland.
- Auf der Achterbahn darf ein Kind jeweils nur in Begleitung eines Erwachsenen fahren.
- Nur Erwachsene/Kinder, die sich kennen, sollen zusammen fahren. Wieviele Kinder können maximal eine Fahrt mitmachen?

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

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 $c(e) = 1$

$\longrightarrow$ Anwendung des Ford-Fulkerson-Algorithmus

Zusammenfassung I

- Viele wichtige Informatikprobleme lassen sich mit gerichteten bzw. ungerichteten Graphen behandeln.
- wesentliche Implementierungsalternativen: Adjazenzmatrix und Adjazenzlisten
- Algorithmen mit linearem Aufwand:
  - Traversierungvon Graphen: Breitensuche vs. Tiefensuche
  - Topologisches Sortieren
  - Test auf Azyklität

Zusammenfassung II

- Weitere wichtige Algorithmen:
  - Warshall-Algorithmus zur Bestimmung der transitiven Hülle
  - Warshall-, Dijkstra- und Bellmann-Ford für kürzeste Wege
  - Kruskal-Algorithmus für minimale Spannbäume
  - Ford-Fulkerson-Algorithmus für maximale Flüsse bzw. maximales Matching
- viele NP-vollständige Optimierungsprobleme
  - TravelingSalesmanProblem, Cliquenproblem, Färbungsproblem ...
  - Bestimmung eines planaren Graphen (Graph-Darstellung ohne überschneidende Kanten)