

Algorithmen und Datenstrukturen 2

Sommersemester 2006
5. Vorlesung

Peter F. Stadler

Universität Leipzig
Institut für Informatik
studla@bioinf.uni-leipzig.de

Wdhlg.: Dijkstra-Algorithmus I

Bestimmung der von einem Knoten ausgehenden kürzesten Wege

- gegeben: kanten-bewerteter Graph $G = (V, E, g)$ mit $g: E \rightarrow \mathbb{R}^+$ (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 $\text{succ}(u)$ liefert die Menge der direkten Nachfolger von u

- Verallgemeinerung der Breitensuche (gewichtete Entfernung)
- funktioniert nur bei nicht-negativen Gewichten
- Optimierung gemäß Greedy-Prinzip

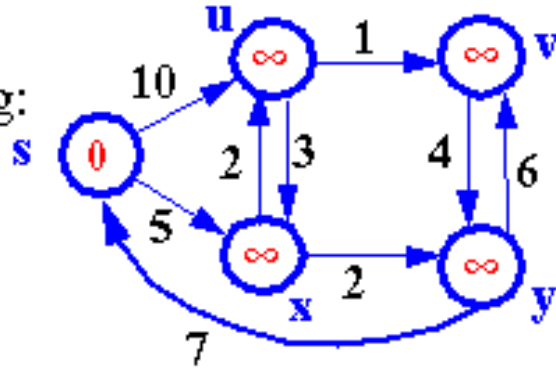
Wdhlg.: Dijkstra-Algorithmus II

Dijkstra (G,s):

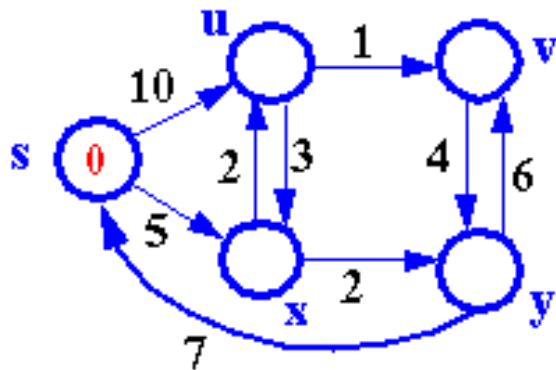
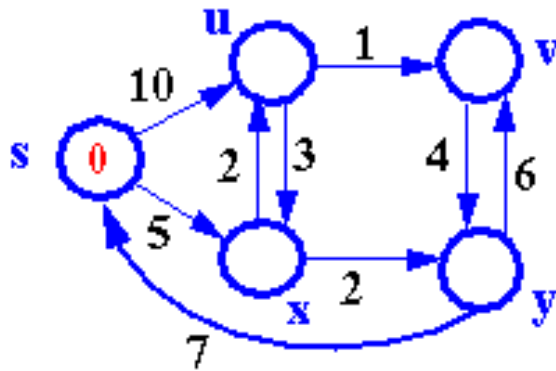
```
for each Knoten  $v \in V - s$  do {  $D[v] = \infty$ ;};  
 $D[s] = 0$ ; PriorityQueue  $Q = V$ ;  
while not isEmpty(Q) do {  $v = \text{extractMinimum}(Q)$ ;  
    for each  $u \in \text{succ}(v) \cap Q$  do {  
        if  $D[v] + g((v,u)) < D[u]$  then  
            {  $D[u] = D[v] + g((v,u))$ ;  
              adjustiere  $Q$  an neuen Wert  $D[u]$ ; }  
    }  
};
```

Dijkstra-Algorithmus: Beispiel

Initialisierung:



$$Q = \langle (s:0), (u:\infty), (v:\infty), (x:\infty), (y:\infty) \rangle$$



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-Algorithmus mit Start-Knoten u (Beendigung sobald v aus Q entfernt wird)

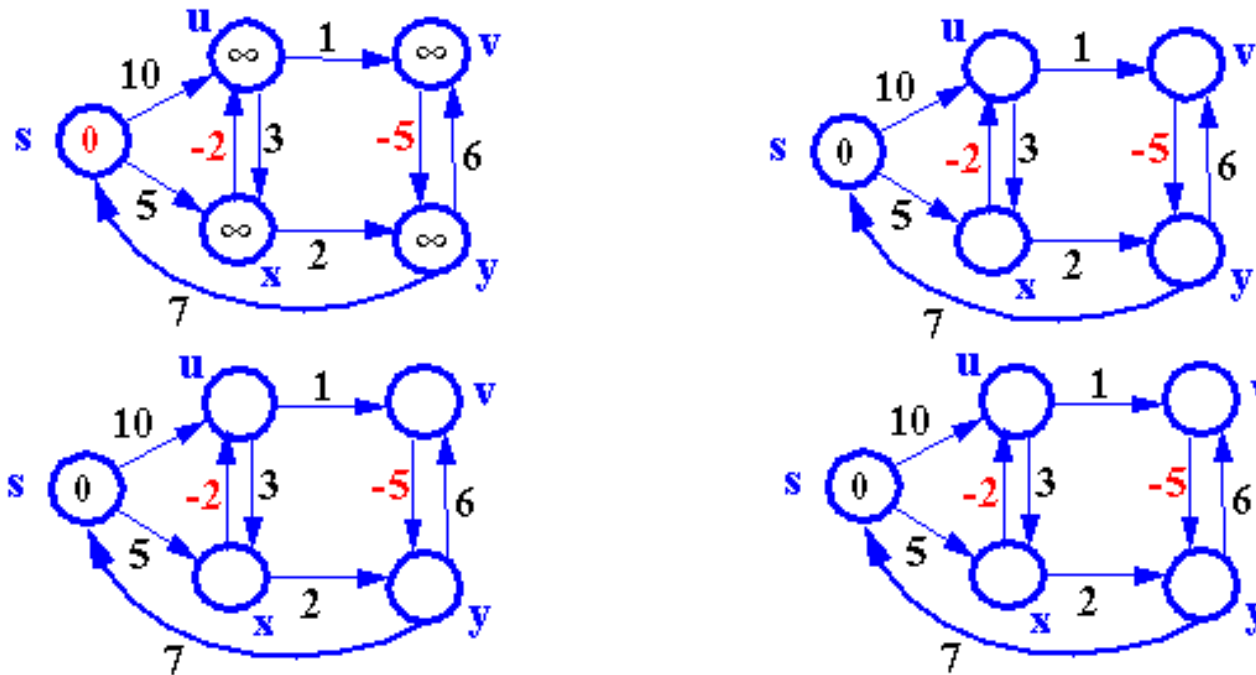
Kürzeste Wege mit negativen Kanten- gewichten (ohne negative Zyklen)

Bellmann-Ford-Algorithmus $BF(G,s)$:

```

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

Beispiel:



Minimale Spann bäume I

Problemstellung: zu zusammenhängendem Graph soll Spannbaum (aufspannender Baum) mit minimalem Kantengewicht (minimale Gesamtlänge) bestimmt werden

- relevant z.B. zur Reduzierung von Leitungskosten in Versorgungsnetzen

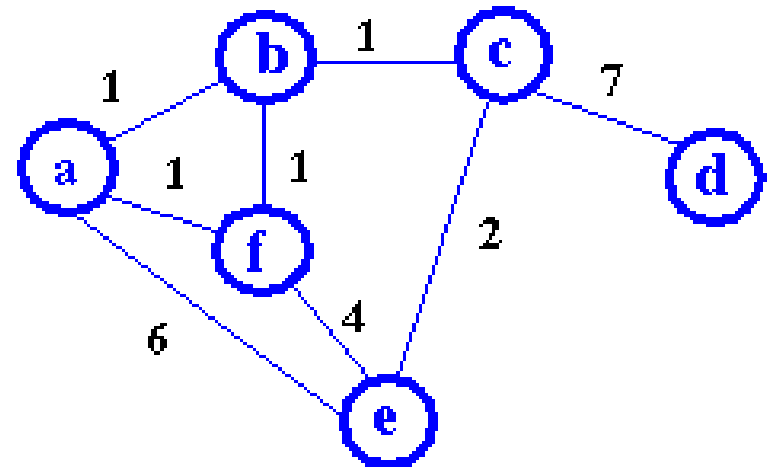
Minimale Spann bäume II

Kruskal-Algorithmus (1956)

- Sei $G = (V, E, g)$ mit $g: E \rightarrow \mathbb{R}$ (Kantengewichte) gegebener ungerichteter, zusammenhängender Graph.

Zu bestimmen minimaler Spannbaum $T = (V, E')$

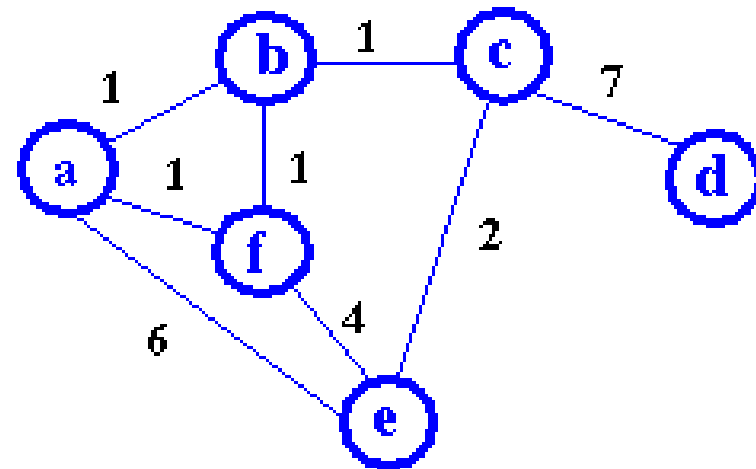
- $E' = \{\}$; sortiere E nach Kantengewicht und bringe die Kanten in PriorityQueue Q ; jeder Knoten v bilde eigenen Spannbaum(-Kandidat)
- solange Q nicht leer:
 - entferne erstes Element $e = (u,v)$
 - wenn beide Endknoten u und v im selben Spannbaum sind, verwerfe e , ansonsten nehme e in E' auf und fasse die Spannbäume von u und v zusammen



Minimale Spann bäume II

Analog: Bestimmung maximaler Spann bäume (absteigende Sortierung)

Anwendung des Kruskal-Algorithmus

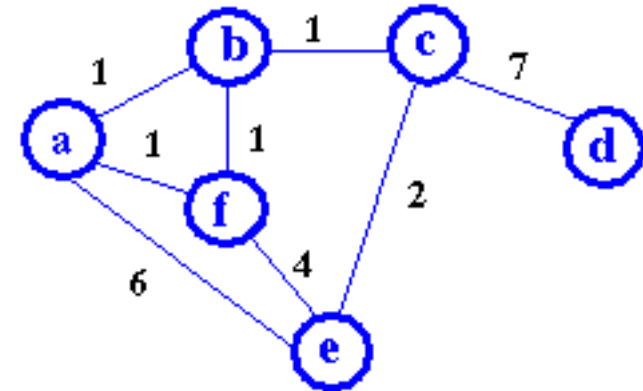


Komplexität $O(m \log n)$

Minimale Spann bäume III

Alternative Berechnung (Dijkstra)

- Startknoten s
- Knotenmenge B enthält bereits abgearbeitete Knoten



s = an Kante mit minimalem Gewicht beteiligter Knoten

B={ s }; E' = { };

while |B| < |V| do {

wähle (u,v) ∈ E mit minimalem Gewicht mit u ∈ B, v ∉ B;

füge (u,v) zu E' hinzu;

füge v zu B hinzu; }

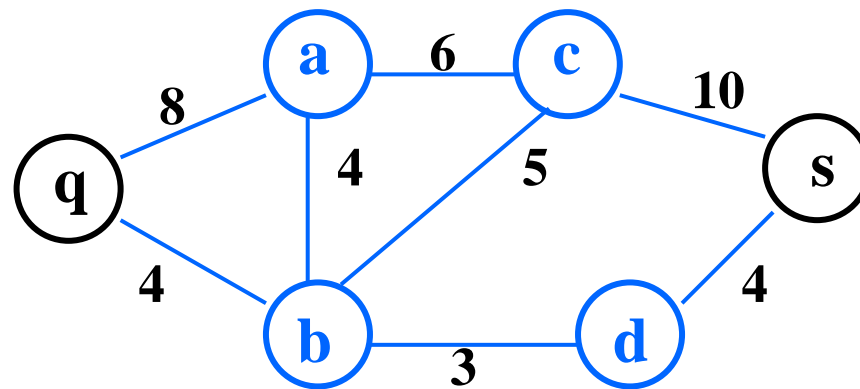
Es wird nur 1 Spannbaum erzeugt

Effiziente Implementierbarkeit mit PriorityQueue über Kantengewicht

Flüsse in Netzwerken I

Anwendungsprobleme:

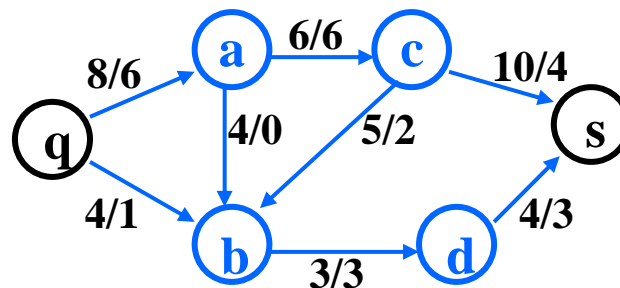
- Wieviel Autos können durch ein Straßennetz fahren?
- Wieviel Abwasser fasst ein Kanalnetz?
- Wieviel Strom kann durch ein Leitungsnetz fließen?



Kantenmarkierung:
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 $c: E \rightarrow \mathbb{Z}^+$.
- Ein Fluß für das Netzwerk ist eine Funktion $f: E \rightarrow \mathbb{Z}$, so daß gilt:
 - Kapazitätsbeschränkung: $f(e) \leq c(e)$, für alle e in E .
 - Flußerhaltung: für alle 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: $\sum_{(q,v) \in E} f((q,v))$



Kantenmarkierung:
Kapazität $c(e)$ / Fluß $f(e)$

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

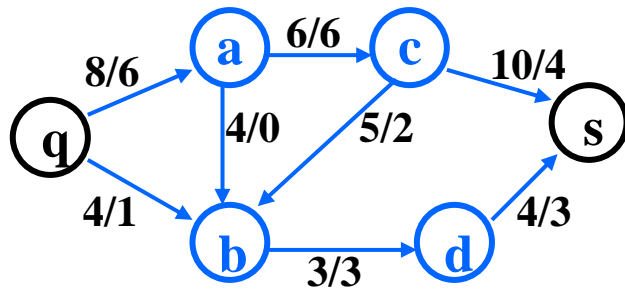
Restgraph

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

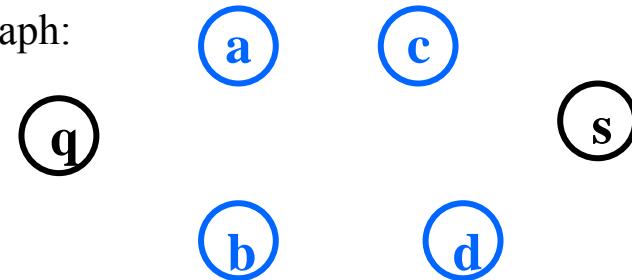
- Wir definieren die Restkapazität einer Kante $e = (v,w)$ wie folgt:

$$\text{rest}(e) = \begin{cases} c(e) - f(e) & \text{falls } e \in E \\ f((w,v)) & \text{falls } (w,v) \in E \end{cases}$$

- Der Restgraph von f (bzgl. G) besteht aus den Kanten $e \in E'$, für die $\text{rest}(e) > 0$



Restgraph:

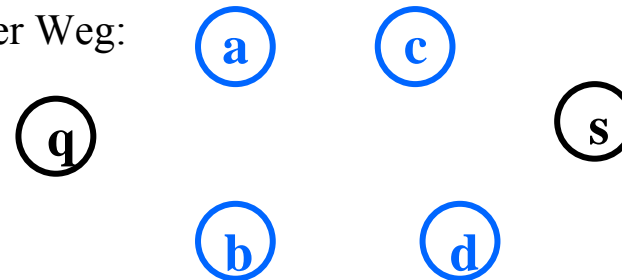


Kantenmarkierung: $\text{rest}(e)$

Zunehmender Weg

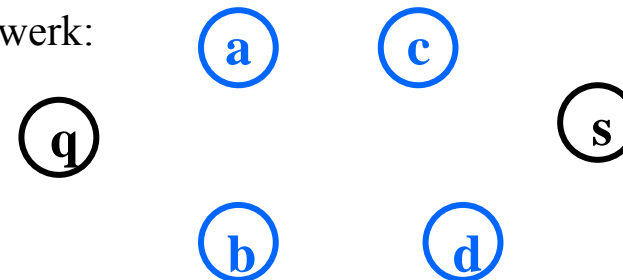
- Jeder gerichtete Pfad von q nach s im Restgraphen heißt zunehmender Weg.

zunehmender Weg:



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

Optimiertes Netzwerk:



Min-Cut-Max-Flow-Theorem

Theorem (Min-Cut-Max-Flow-Theorem):

Sei f zulässiger Fluß für G . Folgende Aussagen sind 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 .

Ford-Fulkerson-Algorithmus

Ford-Fulkerson-Algorithmus

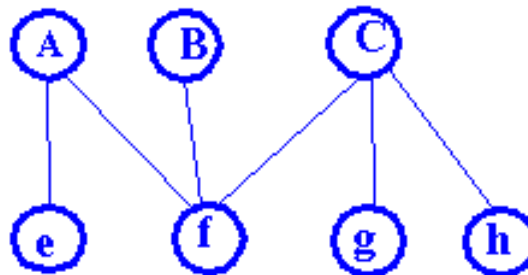
- füge solange zunehmende Wege zum Gesamtfluß hinzu wie möglich
- Kapazität erhöht sich jeweils um Minimum der verfügbaren Restkapazität der einzelnen Kanten des zunehmenden Weges

```
for each  $e \in E$  {  $f(e) = 0$ ; }  
while (es gibt zunehmenden Weg  $p$  im Restgraphen von  $G$ )  
{  
   $r = \min(\text{rest}(e) \mid e \text{ liegt auf } p)$ ;  
  for each  $e = (v, w)$  auf  $p$   
  {  
    if ( $e \in E$ )  $f(e) = f(e) + r$ ;  
    else  $f((w, v)) = f((w, v)) - r$ ;  
  }  
}
```

Maximales 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 $|M| < |M'|$

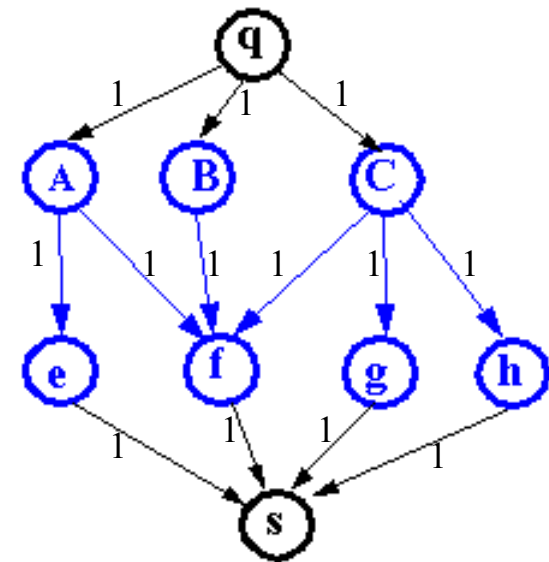
Ein bipartiter Graph ist ein Graph, dessen Knotenmenge V in zwei disjunkte Teilmengen V_1 und V_2 aufgeteilt ist, und dessen Kanten jeweils einen Knoten aus V_1 mit einem aus V_2 verbinden

Matching und maximaler Fluß

Maximales Matching kann auf maximalen Fluß zurückgeführt werden:

- Quelle und Senke hinzufügen.
- Kanten von $V1$ nach $V2$ richten.
- Jeder Knoten in $V1$ erhält eingehende Kante von der Quelle.
- Jeder Knoten in $V2$ erhält ausgehende Kante zur Senke.
- Alle Kanten erhalten Kapazität $c(e) = 1$.

→ Anwendung des Ford-Fulkerson-Algorithmus



Kantenmarkierung: Kapazität $c(e)$

Zusammenfassung I

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

Zusammenfassung II

- **Weitere wichtige Algorithmen:**

- Warshall-Algorithmus zur Bestimmung der transitiven Hülle
- Dijkstra-Algorithmus bzw. Bellmann-Ford für kürzeste Wege
- Kruskal-Algorithmus für minimale Spannbäume
- Ford-Fulkerson-Algorithmus für maximale Flüsse bzw. maximales Matching

viele NP-vollständige Optimierungsprobleme

- Traveling Salesman Problem, Cliquesproblem, Färbungsproblem ...
- Bestimmung eines planaren Graphen (Graph-Darstellung ohne überschneidende Kanten)