5.4 Gerichtete Graphen

Kanten besitzen eine **Richtung**. Die Kante A-B ist nicht mit der Kante B-A identisch.

Fragen:

- Gibt es einen gerichteten Weg zwischen zwei Knoten?
- Gibt es einen Rückweg?
- Welche Knoten sind von einem bestimmten Ausgangsknoten aus erreichbar?

Antwort:

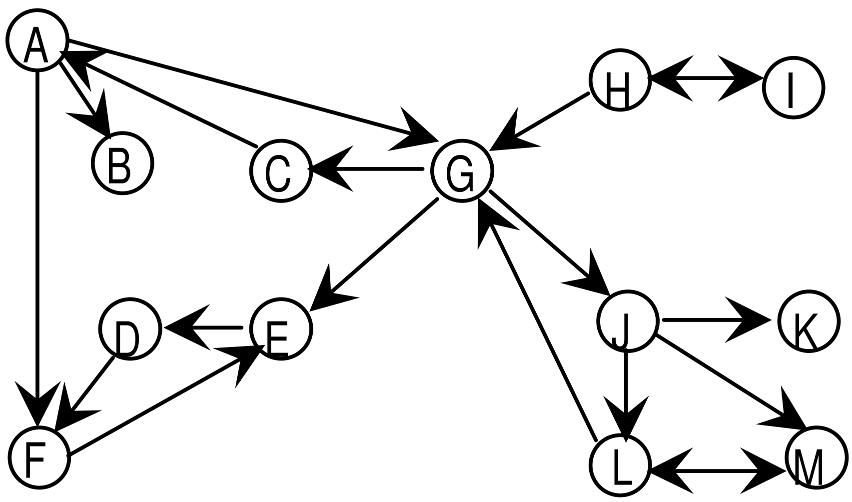
Algorithmus Tiefensuche

=> Verwendung des gleichen Algorithmus wie beim ungerichteten Graphen.

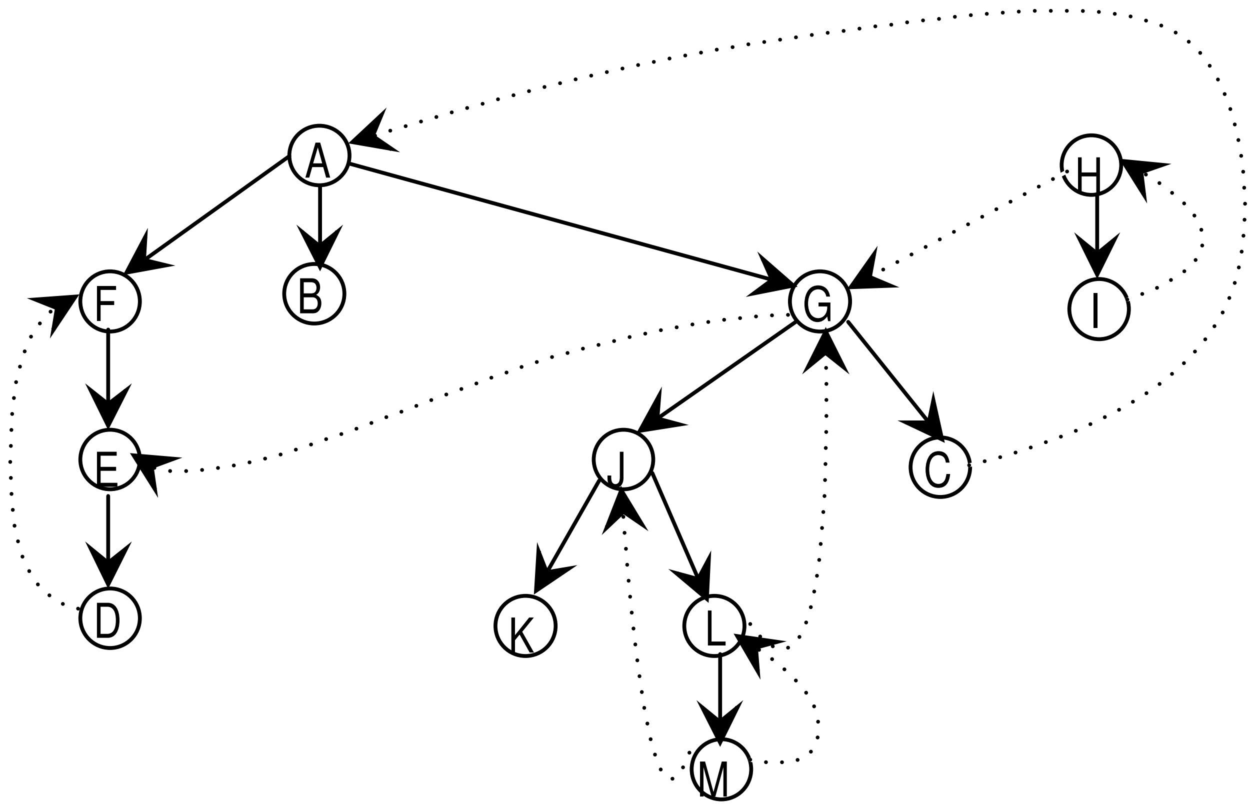
Unterschied zur Tiefensuche in ungerichteten Graphen:

=> **Welche Knoten erreichbar sind, hängt vom Ausgangsknoten ab**. Deshalb ist das Problem der Erreichbarkeit von einem bestimmten Startknoten aus nicht identisch mit dem Problem des Zusammenhangs von ungerichteten Graphen.

Gerichteter Graph, Beispiel



Tiefensuch- Wald für diesen Graphen



5.4.1 Transitive Hülle eines Graphen

- Welche Knoten von einem bestimmten Knoten aus erreicht werden können; ist besonders einfach zu sehen, wenn wir zu allen erreichbaren Knoten eine direkte Kante explizit konstruieren.
- Ein Graph, der dies darstellt, heißt die **transitive Hülle** des ursprünglichen Graphen.
- Die transitive Hülle ist i.a. ein **dichter** Graph.
- Berechnung:
  - für lichte Graphen: mit Tiefensuche von jedem Knoten aus
  - für dichte Graphen:mit Warshall's Algorithmus (1962)
- **Tiefensuche**:
  - Muß von **jedem** Knoten aus explizit gestartet werden
  - Jeder erreichte Knoten wird mit dem Ausgangsknoten verbunden.

Programm

```
for (k = 1; k <= V; k++)
  {
    id = 0;
    for (j = 1; j <= V; j++) val[j] = 0;
    visit[k];
    printf("\n");
  }
```

Ausgabe

A	F	E	D	B	G	J	K	L	M	C		
B												
C	A	F	E	D	B	G	J	K	L	M		
D	F	E										
E	D	F										
F	E	D										
G	J	K	L	M	C	A	F	E	D	B		
H	G	J	K	L	M	C	A	F	E	D	B	I
I	H	G	J	K	L	M	C	A	F	E	D	B
J	K	L	G	C	A	F	E	D	B	M		
K												
L	G	J	K	M	C	A	F	E	D	B		
M	L	G	J	K	C	A	F	E	D	B		

- Zahl der Operationen :   0 ((V * (V+E))

Warshall's Algorithmus

Idee:
Wenn es eine Kante von Knoten x zu Knoten y gibt und eine Kante von Knoten y zu Knoten j, so wird eine Kante von Knoten x zu Knoten y addiert. Wiederholung, bis keine neuen Kanten mehr addiert werden.

```
/* mit Adjazenzmatrix */
for (y = 1; y <= V; y++)
  for (x = 1; x <= V; x++)
    if (a[x][y])
      for (j = 1; j <= V; j++)
        if (a[y][j] a [x][j] = 1;
```

Der "Trick" von Warshall

Falls ein Weg existiert, um von Knoten `x` nach Knoten `y` zu gelangen, *unter ausschließlicher Verwendung von Knoten mit Indizes, die kleiner als* `y` *sind*, und ein Weg, um von Knoten `y` nach Knoten `j` zu gelangen, so existiert auch ein Weg, um von Knoten `x` nach Knoten `j` zu gelangen, *unter ausschließlicher Verwendung von Knoten mit Indizes, die kleiner als* `y+1` *sind*. Das obige Programm ist eine direkt Implementation dieser Aussage.

Die Funktionsweise des Algorithmus von Warshall

Mit dem Verfahren von Warshall wird die Adjazenzmatrix eines Graphen in die Adjazenzmatrix seiner transitiven Hülle umgewandelt. Eine Möglichkeit, dem Algorithmus zu folgen, besteht darin, sich ihn in der Weise vorzustellen, daß jedesmal in einer ganzen Zeile der Matrix auf einmal Werte gesetzt werden. Die Operation für Spalte *y* besteht darin, jede Zeile mit einer Eins in Spalte *y* durch das Ergebnis der OR-Verknüpfung dieser Zeile und *Zeile y* zu ersetzen.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
A	1	1	0	0	0	1	1	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0	0
C	1	0	1	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	0	1	0	0	0	0	0	0	0
E	0	0	0	1	1	0	0	0	0	0	0	0	0
F	0	0	0	0	1	1	0	0	0	0	0	0	0
G	0	0	1	0	1	0	1	0	0	1	0	0	0
H	0	0	0	0	0	0	1	1	1	0	0	0	0
I	0	0	0	0	0	0	0	1	1	0	0	0	0
J	0	0	0	0	0	0	0	0	0	1	1	1	1
K	0	0	0	0	0	0	0	0	0	0	1	0	0
L	0	0	0	0	0	0	1	0	0	0	0	1	1
M	0	0	0	0	0	0	0	0	0	0	0	1	1

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
A	1	1	0	0	0	1	1	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0	0
C	1	0	1	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	0	1	0	0	0	0	0	0	0
E	0	0	0	1	1	0	0	0	0	0	0	0	0
F	0	0	0	0	1	1	0	0	0	0	0	0	0
G	0	0	1	0	1	0	1	0	0	1	0	0	0
H	0	0	0	0	0	0	1	1	1	0	0	0	0
I	0	0	0	0	0	0	0	1	1	0	0	0	0
J	0	0	0	0	0	0	0	0	0	1	1	1	1
K	0	0	0	0	0	0	0	0	0	0	1	0	0
L	0	0	0	0	0	0	1	0	0	0	0	1	1
M	0	0	0	0	0	0	0	0	0	0	0	1	1

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**

A	1	1	0	0	0	1	1	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0	0
C	1	0	1	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	0	1	0	0	0	0	0	0	0
E	0	0	0	1	1	0	0	0	0	0	0	0	0
F	0	0	0	0	1	1	0	0	0	0	0	0	0
G	0	0	1	0	1	0	1	0	0	1	0	0	0
H	0	0	0	0	0	0	1	1	1	0	0	0	0
I	0	0	0	0	0	0	0	1	1	0	0	0	0
J	0	0	0	0	0	0	0	0	0	1	1	1	1
K	0	0	0	0	0	0	0	0	0	0	1	0	0
L	0	0	0	0	0	0	1	0	0	0	0	1	1
M	0	0	0	0	0	0	0	0	0	0	0	1	1

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
A	1	1	0	0	0	1	1	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0	0
C	1	0	1	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	0	1	0	0	0	0	0	0	0
E	0	0	0	1	1	0	0	0	0	0	0	0	0
F	0	0	0	0	1	1	0	0	0	0	0	0	0
G	0	0	1	0	1	0	1	0	0	1	0	0	0
H	0	0	0	0	0	0	1	1	1	0	0	0	0
I	0	0	0	0	0	0	0	1	1	0	0	0	0
J	0	0	0	0	0	0	0	0	0	1	1	1	1
K	0	0	0	0	0	0	0	0	0	0	1	0	0
L	0	0	0	0	0	0	1	0	0	0	0	1	1
M	0	0	0	0	0	0	0	0	0	0	0	1	1

Algorithmus von Warshall, Komplexität

Der Algorithmus von Warshall ermöglicht die Berechnung der transitiven Hülle in O (V^3) Schritten.

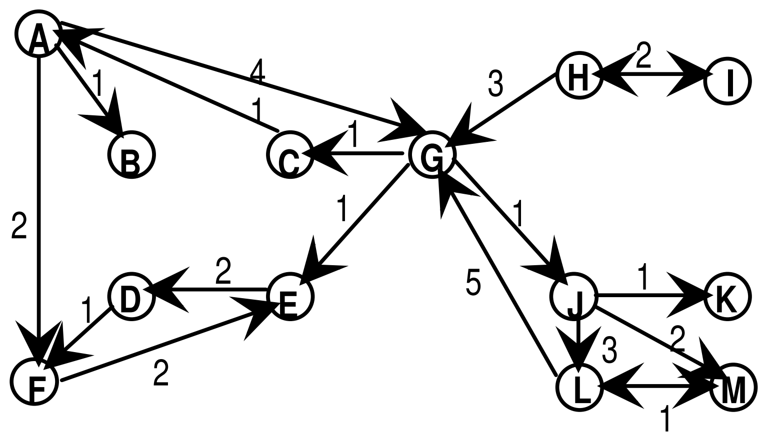
Alle kürzesten Pfade: Algorithmus von Floyd

Für gewichtete Graphen (gerichtet oder nicht) ist es manchmal wünschenswert, eine Tabelle anzugeben, die es gestattet, den kürzesten Pfad von *x* nach *y* für alle Paare von Knoten zu finden. Dies ist das *Problem des kürzesten Pfades für alle Paare.*
Wie oben findet der Algorithmus des kürzesten Pfades aus dem vorangegangenen Kapitel den kürzesten Pfad vom Anfangsknoten zu jedem anderen Knoten, so daß wir nur diese Prozedur *V* mal anwenden müssen, beginnend in jedem Knoten. Dies ergibt einen Algorithmus, der in *O((E+V)V log V)* Schritten abläuft. Doch es ist auch möglich, ein Verfahren anzuwenden, das mit dem Verfahren von Warshall vergleichbar ist und gewöhnlich R.W. Floyd zugeschrieben wird:

```
for (y = 1; y <= V; y++)
 for (x = 1; x <= V; x++)
  if (a[x][y])
   for (j = 1; j <= V; j++)
    if (a[y][j] > 0)
     if (!a[x][j] || (a[x][y]+a[y][j] < a[x][j]))
      a[x][j] = a[x][y] + a[y][j];
```

Die Struktur des Algorithmus ist genau die gleiche wie beim Verfahren von Warshall. Anstelle der Verwendung von OR zur Verfolgung der Pfade führen wir für jede Kante eine kleine Berechnung aus, um zu bestimmen, ob sieTeil eines neuen kürzesten Pfades ist: "Der kürzeste Weg von Knoten *x* nach Knoten *j unter ausschließlicher Benutzung von Knoten mit Indizes, die kleiner als y sind,* oder, wenn er kürzer ist, der kpürzeste Weg von *x* nach *y* plus dem Abstand zwischen *y* und *j."*

Algorithmus von Floyd, Beispiel



	A	B	C	D	E	F	G	H	I	J	K	L	M
A	0	1	0	0	0	2	4	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0
C	1	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	1	0	0	0	0	0	0	0
E	0	0	0	2	0	0	0	0	0	0	0	0	0
F	0	0	0	0	2	0	0	0	0	0	0	0	0
G	0	0	1	0	1	0	0	0	0	1	0	0	0
H	0	0	0	0	0	0	3	0	1	0	0	0	0
I	0	0	0	0	0	0	0	1	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	1	3	2
K	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	5	0	0	0	0	0	1
M	0	0	0	0	0	0	0	0	0	0	0	1	0

	A	B	C	D	E	F	G	H	I	J	K	L	M
A	0	1	0	0	0	2	4	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0
C	1	2	0	0	0	3	5	0	0	0	0	0	0
D	0	0	0	0	0	1	0	0	0	0	0	0	0
E	0	0	0	2	0	0	0	0	0	0	0	0	0
F	0	0	0	0	2	0	0	0	0	0	0	0	0
G	0	0	1	0	1	0	0	0	0	1	0	0	0
H	0	0	0	0	0	0	3	0	1	0	0	0	0
I	0	0	0	0	0	0	0	1	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	1	3	2
K	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	5	0	0	0	0	0	1
M	0	0	0	0	0	0	0	0	0	0	0	1	0

	A	B	C	D	E	F	G	H	I	J	K	L	M
A	6	1	5	6	4	2	4	0	0	5	6	8	7
B	0	0	0	0	0	0	0	0	0	0	0	0	0
C	1	2	6	7	5	3	5	0	0	6	7	9	8
D	0	0	0	5	3	1	0	0	0	0	0	0	0
E	0	0	0	2	5	3	0	0	0	0	0	0	0
F	0	0	0	4	2	5	0	0	0	0	0	0	0
G	2	3	1	3	1	4	6	0	0	1	2	4	
H	5	6	4	6	4	7	3	2	1	4	5	7	6
I	6	7	5	7	5	8	4	1	2	5	6	8	7
J	0	0	0	0	0	0	0	0	0	0	1	2	3
K	0	0	0	0	0	0	0	0	0	0	0	0	0
L	7	8	6	8	6	9	5	0	0	6	7	9	1
M	0	0	0	0	0	0	0	0	0	0	0	1	0

	A	B	C	D	E	F	G	H	I	J	K	L	M
A	6	1	5	6	4	2	4	0	0	5	6	8	7
B	0	0	0	0	0	0	0	0	0	0	0	0	0
C	1	2	6	7	5	3	5	0	0	6	7	9	8
D	0	0	0	5	3	1	0	0	0	0	0	0	0
E	0	0	0	2	5	3	0	0	0	0	0	0	0
F	0	0	0	4	2	5	0	0	0	0	0	0	0
G	2	3	1	3	1	4	6	0	0	1	2	4	
H	5	6	4	6	4	7	3	2	1	4	5	7	6
I	6	7	5	7	5	8	4	1	2	5	6	8	7
J	10	11	9	11	9	12	8	0	0	9	1	3	2
K	0	0	0	0	0	0	0	0	0	0	0	0	0
L	7	8	6	8	6	9	5	0	0	6	7	9	1
M	8	9	7	9	7	10	6	0	0	7	8	1	2

Algorithmus von Floyd, Komplexität

Der Algorithmus von Floyd ermöglicht die Lösung des Problems des kürzesten Pfades für alle Paare in O (V^3) Schritten.

Gerichtete azyklische Graphen (dags)

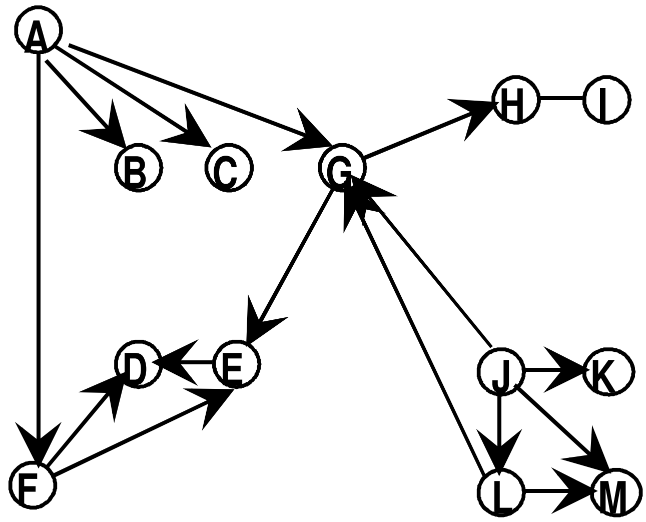
Ein gerichteter Graph frei von gerichteten Zyklen heißt *gerichteter azyklischer Graph* (directed acyclic graph, dag).

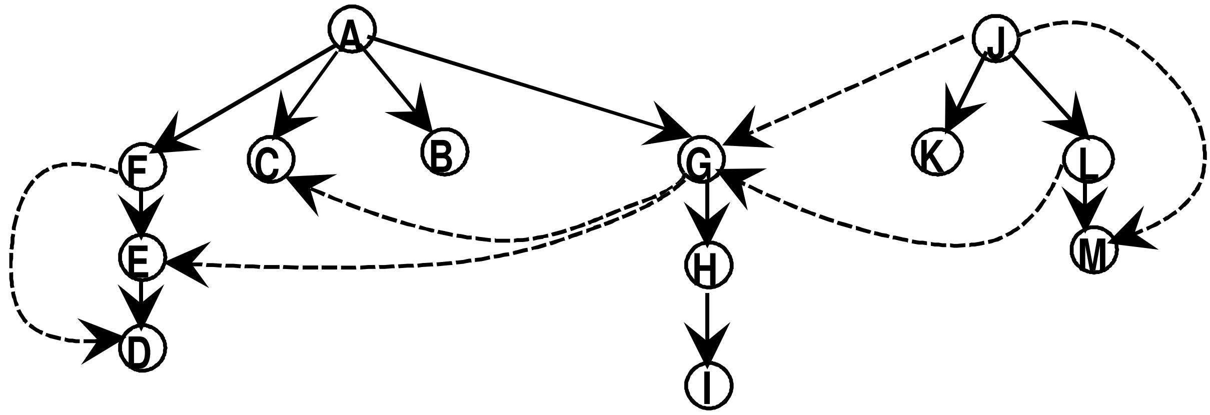
In der Praxis treten gerichtete, azyklische Graphen häufig auf, um Präzedenzbeziehungen zu beschreiben (A muß vor B kommen).

Beispiel

Wartegraph in der Sperrverwaltung eines Betriebssystems, Datenbank-systems, oder Transaktionssystems. Enthält der Wartegraph einen Zyklus, so liegt ein Deadlock vor.

dag, Beispiel





Dags unterscheiden sich tatsächlich sehr stark von allgemeinen gerichteten Graphen; in gewissem Sinne sind sie teils Baum, teils Graph. Bei ihrer Verarbeitung läßt sich ihre spezielle Struktur ausnutzen. Von einem beliebigen Knoten aus betrachtet sieht ein dag wie ein Baum aus; anders gesagt, der Tiefensuch-Wald für eine dag hat keine nach oben führenden Kanten. Die untere Abbildung zeigt den Tiefensuch-Wald, der die Arbeitsweise von *dfs* bei Anwendung auf den dag in der oberen Abbildung beschreibt.

Topologisches Sortieren

Eine grundlegende Operation mit dags ist die Verarbeitung der Knoten des Graphen in einer solchen Reihenfolge, daß kein Knoten vor einem auf ihn zeigenden Knoten verarbeitet wird. Die Knoten im obigen Graphen könnten zum Beispiel in der folgenden Reihenfolge verarbeitet werden:

J K L M A G H I F E D B C

Zeichnet man zu den in diesen Positionen angeordneten Knoten die Kanten, so verlaufen sie alle von links nach rechts. Wie bereits erwähnt, gibt es hierfür offensichtliche Anwendungen, zum Beispiel bei Graphen, die Fertigungsprozesse darstellen. Diese Operation wird *topologisches Sortieren* genannt.

Im allgemeinen führt das topologische Sortieren nicht zu einer eindeutigen Lösung. Zum Beispiel ist die Reihenfolge

A J G F K L E M B H C I D

eine weitere zulässige topologische Ordnung für unser Beispiel (und es gibt viele andere). Bei der erwähnten Anwendung in der Fertigung tritt diese Situation auf, wenn ein Arbeitsgang nicht direkt oder indirekt von einem anderen abhängt und daher in dieser oder jener Reihenfolge ausgeführt werden kann.

Definition Topologische Ordnung

Ein gerichteter Graph kann stets als eine binäre Relation angesehen werden; ein zyklenfreier gerichteter Graph beschreibt also eine Halbordnung.

Eine topologische Sortierung eines gerichteten Graphen ist nun eine vollständige Ordnung über den Knoten des Graphen, die mit der durch die Pfeile ausgedrückten partiellen Ordnung verträglich ist. Genauer:

Eine topologische Sortierung eines gerichteten Graphen $G = (V,E)$ ist eine Abbildung *ord*: $V \rightarrow \{1,...,n\}$ mit $n = |V|$, so daß mit $(v,w) \in E$ auch $ord(v) < ord(w)$ gilt.

Algorithmus: Topologisches Sortieren

- Suche einen Knoten, der keine Nachkommen hat.
- Entferne ihn aus dem Graphen und setze ihn an das Ende der topologisch geordneten Liste.
- Falls der Graph nicht leer ist, beginne von vorne.

Oder:

- Suche einen Knoten, der keine Vorgänger hat.
- Entferne ihn aus dem Graphen und setze ihn an den Anfang der topologische geordneten Liste.
- Falls der Graph nicht leer ist, beginne von vorne.

Umgekehrtes topologisches Sortieren

Gelegentlich ist es nützlich, die Kanten in einem Graphen auf umgekehrte Weise zu definieren, indem man sagt, daß eine von *x* nach *y* gerichtete Kante bedeutet, daß Knoten *x* von Knoten *y* "abhängt".

Dies entspricht der Anordnung der Knoten in einer Zeile in der Weise, daß alle Kanten von rechts nach links verlaufen. Eine *umgekehrte topologische Reihenfolge* für den Graph aus unserem Beispiel ist:

D E F C B I H G A K M L J

Der Unterschied ist hierbei nicht von größerer Bedeutung: Die Ausführung eines umgekehrten topologischen Sortierens für einen Graph ist äquivalent zur Ausführung eines topologischen Sortierens für den Graph, den man durch Umkehrung der Richtung aller Kanten erhält.

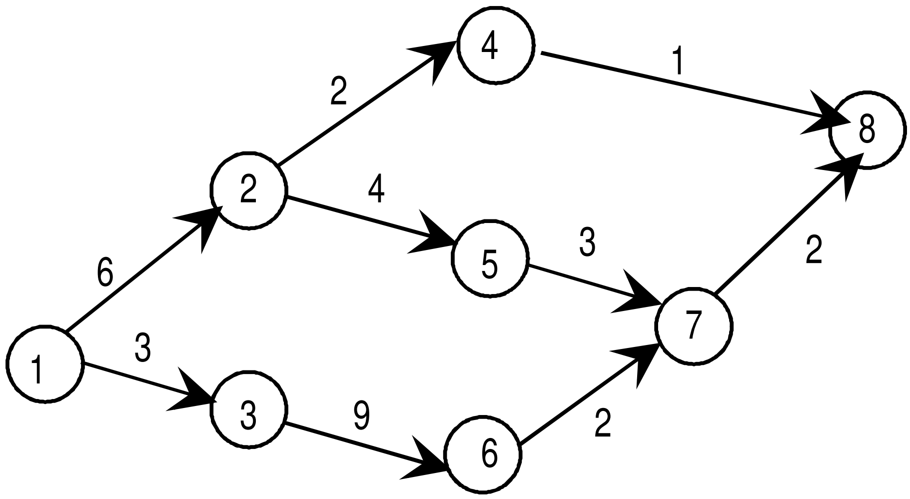
Wir haben schon eine Algorithmus für das umgekehrte topologische Sortieren kennengelernt: Die standardmäßige rekursive Tiefensuchprozedur! Wenn der eingegebene Graph ein dag ist, bewirkt eine einfache Änderung von *visit* in der Weise, daß vor dem *Verlassen* der Prozedur der gerade besuchte Knoten ausgegeben wird (zum Beispiel durch Einfügen von *printf("%c", name[k])* unmittelbar am Ende), daß *dfs* die Knoten in der umgekehrten topologischen Reihenfolge ausgibt.

Die Methode des Kritischen Pfades (CPM)

(die Brücke zu den betrieblichen Anwendungen)

Man kann einen gerichteten, gewichteten, azyklischen Graphen als Aktivitätsgraphen auffassen. Die Gewichte entsprechen Zeitdauern, ein Pfeil von A nach B besagt, daß Ereignis A abgeschlossen sein muß, bevor Ereignis B erreicht werden kann.

Beispiel



Es gibt eine Reihe von Techniken für Aktivitätsgraphen:

CPM: Critical Path Method

PERT: Performance Evaluation und Review Technique

und viele mehr.

Die Projektstruktur als Graph zeigt auf, daß einige Aufgaben parallel ausgeführt werden können. Die Aufgaben der Ereignisse 2 und 3 beispielsweise können gleichzeitig ausgeführt werden. Interessant für das Projektmanagement ist die Bestimmung der Ereignisse, die kritisch für die fristgerechte Abwicklung eines Projektes sind. Ein Ereignis ist *kritisch*, wenn eine Verzögerung deren geplanten Abwicklungszeit ebenfalls die verzögerte Abwicklung des Gesamtprojektes bedeutet. Die kürzeste Abwicklungszeit eines gesamten Projektes entspricht dem längsten Pfad durch den Graph. Der längste Pfad heißt *kritischer Pfad*; entlang diesem Pfad liegen die kritischen Ereignisse. Die Länge dieses Pfades heißt *Kritische Pfadzeit (critical path time, CPT)*. Ein Informationssystem könnte Programme enthalten sowohl zum Auffinden kritischer Pfade und CPT als auch zur Entwicklung und Darstellung von Aktivitätsgraphen.
Eine Möglichkeit, den kritischen Pfad aufzuspüren und die zulässigen Verzögerungszeiten der Ereignisse zu bestimmen, die sich nicht auf dem kritischen Pfad befinden, ist die Feststellung der frühesten und spätesten Startzeiten für diese Ereignisse. Die *früheste Startzeit* für das Ereignis *i*, FSZ_i, ist die frühstmögliche Auslösezeit für das Ereignis. Für den Beispielgraphen gilt:

Ereignis	FSZ
1	0
2	6
3	3
4	8
5	10
6	12
7	14
8	16

Errechnet wird die FSZ_i aus dem längsten Pfad zwischen dem Startknoten und dem Knoten *i*. Beachte, daß Aufgaben der beiden Knoten 5 und 6 vor Auslösung des Ereignisses 7 beendet sein müssen. Formal heißt das:

$$FSZ_i = \max_{j \varepsilon P(i)} \{ FSZ_j + Z(j,i) \} ,$$

wobei *P(i)* die Menge der unmittelbaren Vorgänger vom Knoten *i* ist und Z(j,i) das Gewicht der Kante zwischen den Knoten *j* und *i*.

Die *späteste Startzeit* eines Ereignisses *i*, SSZ_i, ist die spätestmögliche Zeit zu der ein Ereignis *i* ausgelöst werden kann und zu der der Graph noch innerhalb der kritischen Pfadzeit abgearbeitet werden kann. Die Menge der SSZ_i wird rückwärts, vom letzten Ereignis ausgehend, gerechnet. Für den Beispielgraphen gilt:

Ereignis	SSZ_i
1	0
2	13
3	3
4	15
5	11
6	12
7	14
8	16

Formal heiß das:

$$SSZ_i = \min_{j \in S(i)} \{SSZ_j - Z(i.j)\}$$

wobei *S(i)* die Menge der unmittelbaren Nachfolgern des Knotens i ist.

Die Differenz zwischen der SSZ_i und FSZ_i ist die maximal zulässige Verzögerung des Ereignisses i:

Ereignis	SSZ-FSZ
1	0
2	7
3	0
4	7
5	1
6	0
7	0
8	0

Somit stellen die Knoten 1, 3, 6, 7 und 8 die kritischen Ereignisse dar, die sich entlang dem längsten Pfad im Graphen befinden. Um das Projekt fristgerecht beenden zu können, müssen diese Ereignisse besonders sorgfältig kontrolliert werden. Man beachte, daß der kritische Pfad durch den Graphen geändert würde, falls sich ein unkritisches Ereignis länger als sein zulässiges Fenster verzögert.

5.5 Flüsse in Netzwerken

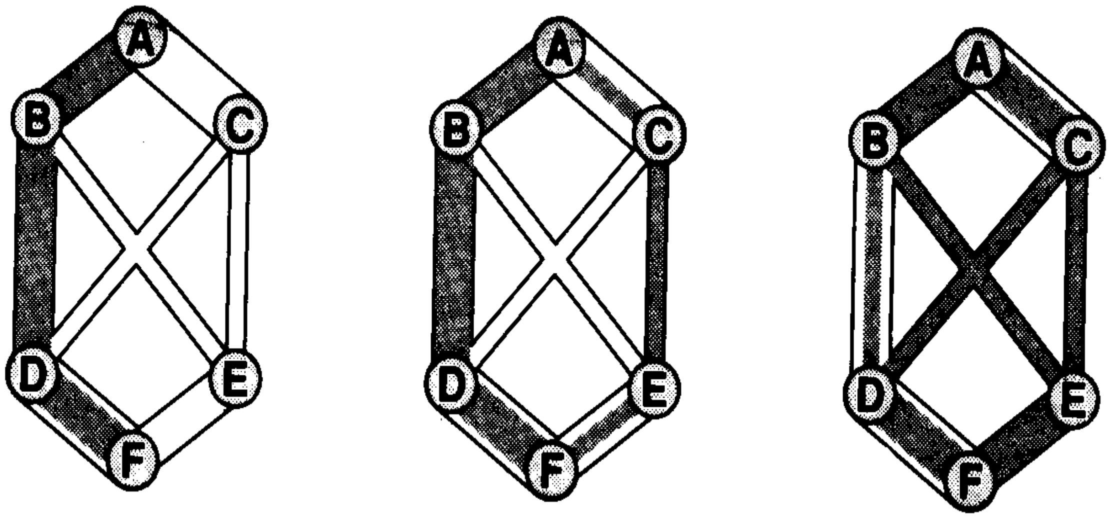
Als einen Spezialfall von gewichteten, gerichteten Graphen betrachten wir Flüsse in Netzwerken. Das Gewicht einer Kante wird als Kapazität einer Rohrleitung interpretiert.

Analog läßt sich auch der Verkehrsfluß auf Straßen oder der Materialfluß in Produktionsprozessen modellieren.

Flüsse in Netzwerken werden im *Operations Research* ausführlich behandelt.

Problemstellung Leitungsnetz

An jeder Verzweigungsstelle gibt es eine Schieber, der den Fluß in die herausführenden Rohre steuert. Im Gleichgewichtszustand fließt "unten" genausoviel Flüssigkeit aus dem Netzwerk, wie "oben" hineinfließt. Ziel ist die Maximierung des Durchflusses durch optimale Einstellung der Schieber.



Formalisierung der Problemstellung

Sei im folgenden G = (V,E) ein gerichteter Graph mit einer Kapazitätsfunktion c: $E \rightarrow IR^+$ (englisch: capacity) und zwei ausgezeichneten Knoten, einer *Quelle* *q* und einer *Senke s*. Unser Ziel ist es, einen maximalen Fluß von *q* nach *s* zu ermitteln. Ein Fluß durch eine Kante muß die Kapazitätsbeschränkung dieser Kante einhalten; an jedem Knoten muß außerdem der Fluß erhalten bleiben, also gleichviel hinein- wie herausfließen (außer an der Quelle und an der Senke). Wir definieren daher einen *Fluß* als eine Funktion *f* : $E \rightarrow IR_o^+$, wobei gilt:

- *Kapazitätsbeschränkung:* für alle $e \in E$ ist $f(e) \leq c(e)$;
- *Flußerhaltung:* für alle $v \in V - \{q,s\}$ ist

$$\sum_{(v',v)\in E} f((v',v)) - \sum_{(v,v'')} f((v,v'')) = 0.$$

Der Wert *w(f)* eines Flusses *f* ist die Summe der Flußwerte aller *q* verlassenden Pfeile, also $w(f) = \sum_{(q,v)\in E} f((q,v)) - \sum_{(v',q)\in E} f((v',q))$.

In unserem Beispiel ist *w(f)* = 3. Ein *maximaler Fluß* in *G* ist ein Fluß *f* in *G* mit maximalem Wert *w(f)* unter allen Flüssen in *G*.

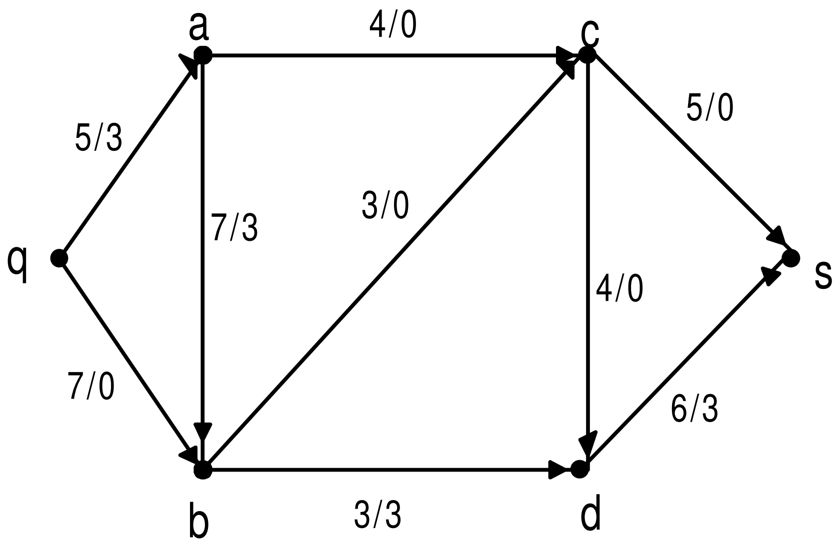
Einfluß der Schieber, Beispiel

Nehmen wir zuerst an, daß der die Leitung AB steuernde Schieber geöffnet ist, wobei er diese Leitung füllt, Leitung BD füllt und DF beinahe füllt, wie es die linke Skizze in der Abbildung zeigt. Nehmen wir dann an, daß Leitung AC geöffnet ist, und daß Schieber C so eingestellt ist, daß er die Leitung CD schließt und Leitung CE öffnet (vielleicht hat der Bediener von Schieber D den Bediener von Schieber C darüber informiert, daß bei ihm aufgrund des von B kommenden Stroms nicht mehr viel zusätzliches Öl passieren kann). Den resultierenden Fluß zeigt die mittlere Skizze der Abbildung: Die Leitungen BD und CE sind voll. Die Durchflußmenge könnte nun noch etwas vergrößert werden, indem über den Pfad ACDF eine Ölmenge geleitet wird, die ausreiche um Leitung DF zu füllen; es existiert jedoch eine bessere Lösung, wie die dritte Skizze zeigt. Durch Umstellung des Schiebers B in der Weise, daß eine Ölmenge umgeleitet wird, die ausreicht, um BE zu füllen, wird in Leitung DF eine genügend große Kapazität frei, die es gestattet, mit Hilfe des Schiebers C Leitung CD vollständig zu öffnen. Die Gesamtmenge, die in das Netzwerk hinein und aus ihm hinaus fließt, läßt sich also erhöhen, indem man die geeigneten Schieberstellungen ermittelt.

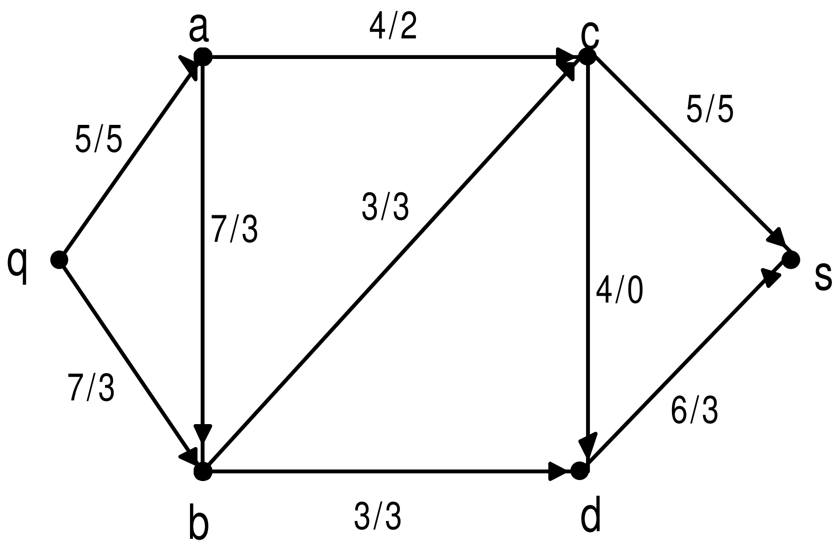
Erste Idee

Erhöhung des Flusses an den nichtgesättigten Kanten.

Beispiel



Erhöhung des Flusses auf Pfad q,a,c,s um 2 und auf Pfad q,b,c,s um 3:



Jetzt ist auf jedem Pfad von q nach s wenigstens eine Kante gesättigt. Der gesamte Fluß ist w = 8. Aber das ist nicht der maximal mögliche Fluß!

Beobachtung: min cut / max flow

Es ist intuitiv klar, daß nicht mehr im Netzwerk fließen kann, als aus der Quelle herausfließt oder in die Senke hineinfließt. In unserem Beispiel verlassen höchstens 12 Einheiten die Quelle, und höchstens 11 fließen in die Senke. Aber nicht nur Quelle und Senke begrenzen den Wert eines maximalen Flusses, sondern jeder Schnitt durch den Graphen, der *q* von *s* trennt.

Definitionen

Ein *(q von s trennender) Schnitt* ist eine Zerlegung der Knotenmenge *V* in zwei Teilmengen *Q* und *S*, so daß *q* zu *Q* und *s* zu *S* gehört.

Die *Kapazität c(Q,S)* eines Schnittes *Q,S* ist die Summe der Kapazitäten von Pfeilen, die von *Q* nach *S* führen.

Ein Schnitt mit kleinster Kapazität unter allen möglichen Schnitten heißt *minimaler Schnitt.*

Schnitte im Beispiel

In dem in der vorletzten Abbildung gezeigten Beispiel ist etwa $Q = \{q,b\}$, $S = \{a,c,d,s\}$ ein Schnitt; die Kapazität $c(Q,S)$ dieses Schnitts ist $c|((q,a))+c((b,c))+c((b,d)) = 11$.

Für einen Fluß f und einen Schnitt Q,S ist der *(netto-) Fluß über den Schnitt*

$$f(q,S) = \sum_{v\in Q, v'\in S, (v,v')\in E} f((v,v')) - \sum_{v\in Q, v'\in S (v',v)\in E} f((v',v)).$$

In unserem Beispiel ist also der Fluß

$$f(\{q,b\},\{a,c,d,s\}) = f((q,a))+f((b,c))+f((b,d))-f((a,b)) = 3+0+3-3=3.$$

Daß dies gerade dem Wert des Flusses $w(f)$ entspricht, ist kein Zufall.

Ganz allgemein gilt für jeden Fluß *f* und jeden Schnitt *Q,S*, daß der Fluß *f(Q,S) = w(f)* ist. Dies sieht man wie folgt ein. Nach Definition ist

$$f(Q,S) = \sum_{\substack{v\in Q,\\ v'\in Q,\\ (v,v')\in E}} f((v,v')) - \sum_{\substack{v\in Q,\\ v\in S,\\ (v',v)\in E}} f((v',v))$$

Addieren wir zur rechten Seite dieser Gleichung

$$\sum_{\substack{v\in Q,\\ v'\in Q,\\ (v,v')\in E}} f((v,v')) - \sum_{\substack{v\in Q,\\ v\in Q,\\ (v,v')\in E}} f((v,v')) + \sum_{\substack{v\in Q,\\ v'\in Q,\\ (v',v)\in E}} f((v',v)) - \sum_{\substack{v\in Q,\\ v'\in Q,\\ (v',v)\in E}} f((v',v))$$

so können wir die Summanden neu zusammenfassen zu

$$f(q,S) = \sum_{\substack{v\in Q,\\ v'\in V,\\ (v,v')\in E}} f((v,v')) - \sum_{\substack{v\in Q,\\ v'\in V,\\ (v',v)\in E}} f((v',v)) + \sum_{\substack{v\in Q,\\ v'\in Q,\\ (v'v)\in E}} f((v',v)) - \sum_{\substack{v\in Q,\\ v'\in Q,\\ (v,v')\in E}} f((v,v'))$$

Wegen der Flußerhaltung ergeben die ersten beiden Summanden zusammen gerade *w(f);* die letzten beiden Summanden ergeben 0, und somit ist die Behauptung nachgewiesen.

Min cut / max flow

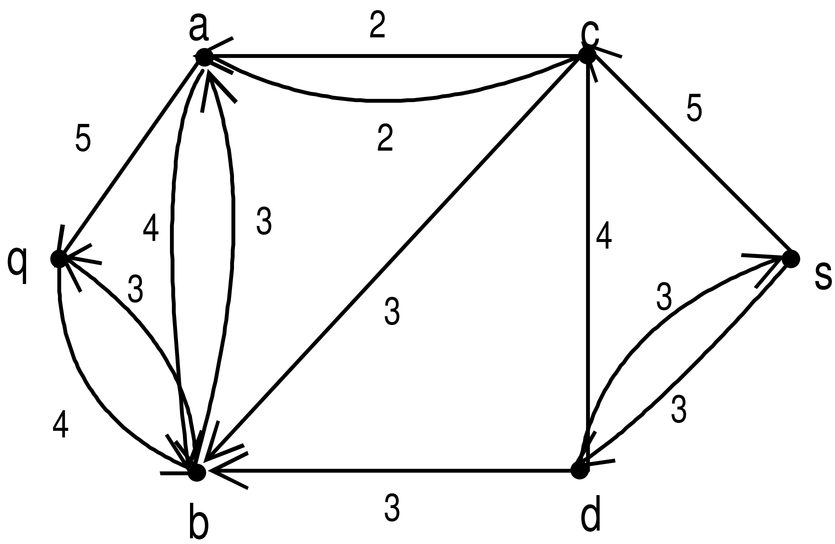
Wegen der Kapazitätsbeschränkung kann man sofort schließen, daß der Fluß über einen beliebigen Schnitt dessen Kapazität nicht übersteigen kann. Damit ist der Wert eines maximalen Flusses sicher nicht größer als die Kapazität eines minimalen Schnittes; wie man leicht beweisen kann, sind in der Tat beide gleich.

Zunehmende Pfade

Wir können das Abändern von Flüssen in Pfaden von *q* nach *s* allgemeiner ausdrücken, wenn wir nicht nur das Erhöhen eines Flusses entlang eines Pfeils mit noch freier *Restkapazität rest(e) = c(e) - f(e)* in Betracht ziehen, sondern auch das Verringern eines Flusses entlang eines entgegengerichteten Pfeiles, also gewissermaßen das Erhöhen eines Flusses entgegen der Pfeilrichtung. Einen Fluß *f(e)* kann man natürlich höchstens um *f(e)* Einheiten verringern; dann ergibt sich für *f(e)* der Wert 0.

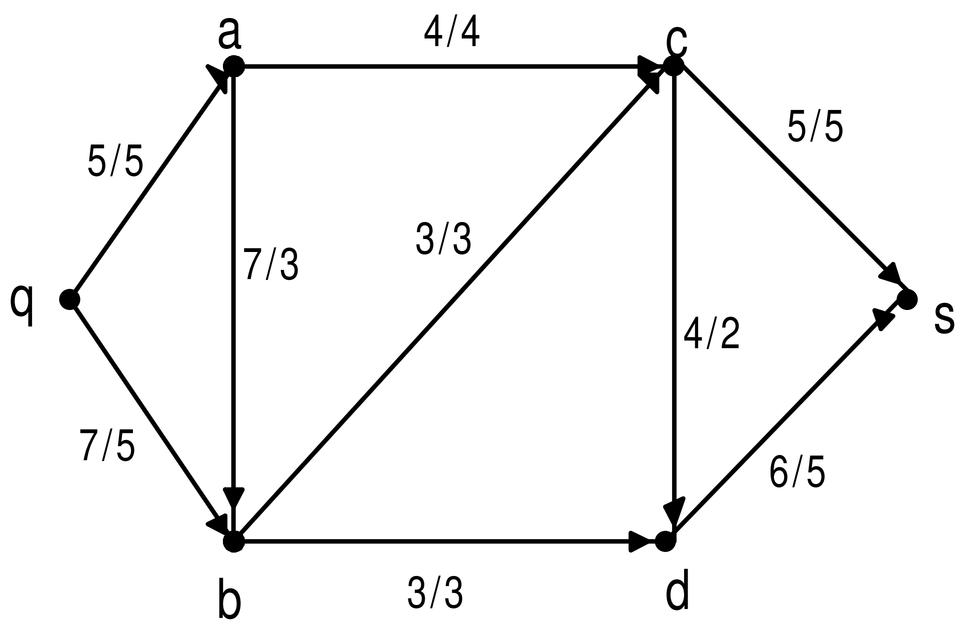
Ein solcher Pfad ohne Rücksicht auf die Pfeilrichtungen (ein *ungerichteter Pfad*) von *q* nach *s* , auf dem man den Fluß erhöhen kann, wird *zunehmender Pfad* genannt. Für jeden Pfeil *e* auf einem zunehmenden Pfad, der in Pfeilrichtung durchlaufen wird (ein *Vorwärtspfeil*), ist $f(e) < c(e)$, also $rest(e) > 0$; für jeden Pfeil, der in Gegenrichtung durchlaufen wird (ein *Rückwärtspfeil*), gilt $f(e) > 0$. Der *Restgraph* zu einem Fluß *f* beschreibt alle Flußvergrößerungsmöglichkeiten: Er enthält einen Pfeil *e*, wenn $rest(e) > 0$ gilt; er enthält den zu *e* entgegengesetzten Pfeil, wenn $f(e) > 0$ gilt.

Restgraph, Beispiel



Jeder Pfad im Restgraphen von *q* nach *s* ist ein zunehmender Pfad für den gegebenen Fluß. In unserem Beispiel ist der einzige zunehmende Pfad der einzige einfache Pfad von *q* nach *s* im Restgraphen, also der Pfad *q,b,a,c,d,s.* Nach der Flußvergrößerung um 2 Einheiten auf diesem Pfad ergibt sich der in der nächsten Abbildung gezeigte Fluß; im zugehörigen Restgraphen führt kein Pfad mehr von *q* nach *s*. Der Fluß hat den Wert 10, ist also maximal. Wir haben im Restgraphen nur solche Pfeile *e* eingezeichnet, für die *rest(e)* > 0 gilt.

Erreichen des maximalen Flusses



Algorithmus von Ford-Fulkerson

Ford und Fulkerson haben 1956 gezeigt, daß ein Fluß *f* genau dann maximal ist, wenn es für *f* keinen zunehmenden Pfad gibt, und daß genau dann der Wert des Flusses *f* der Kapazität des minimalen Schnittes entspricht.
In anderen Worten: Falls jeder von der Quelle zur Senke führende Pfad in einem Netzwerk eine volle Vorwärts-Kante oder eine leere Rückwärts-Kante aufweist, ist der Fluß maximal.

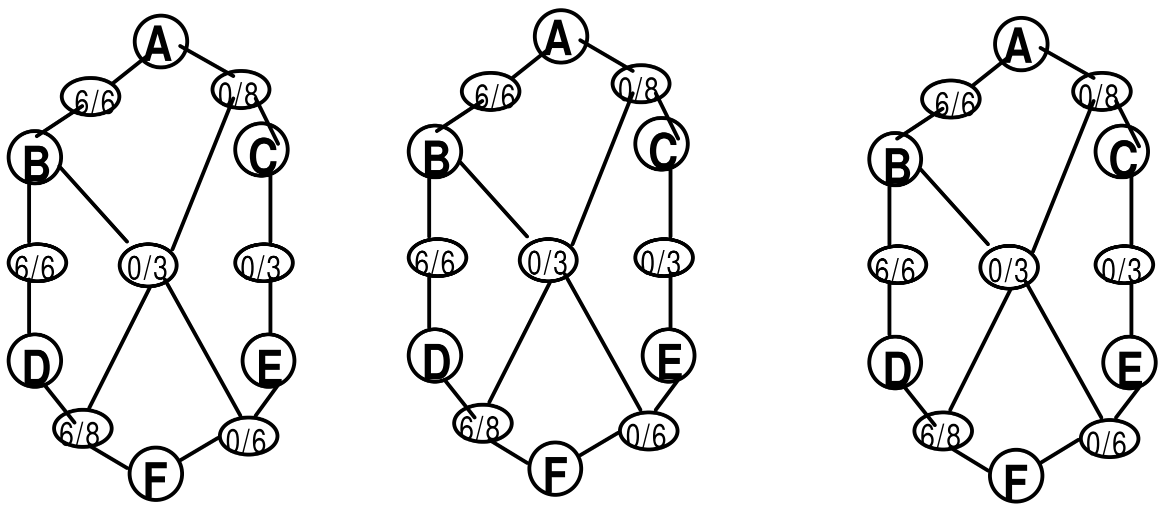
Hieraus ergibt sich unmittelbar der Algorithmus von Ford-Fulkerson:

Algorithmus Maximaler Fluß durch zunehmende Wege

```
{berechnet zu einem gerichteten Graphen G = (V,E)
mit Kapazität c: E→IR⁺
einen maximalen Fluß f : E→IR⁺₀  für G}
begin
  1.   {Initialisiere mit Nullfluß:}
     for all e ∊ E do
        f(e) :=0;
  2.   {iterierte Flußvergrößerung:}
        while es gibt einen zunehmenden Weg p do
          begin
           r:=min{rest(e)|e liegt auf Weg p im
              Restgraphen};
            erhöhe Fluß f auf diesem Pfad p um r
         end
  end{Maximaler Fluß}
```

(für ganzzahlige Kapazitäten)

Algorithmus von Ford-Fulkerson, Beispiel



Betrachten wir einen (nach unten) gerichteten Pfad durch das Netzwerk (von der Quelle zur Senke). Es ist klar, daß der Fluß wenigstens um den kleinsten Betrag der ungenutzten Kapazität jeder Kante des Pfades erhöht werden kann, indem der Fluß in allen Kanten des Pfades um diesen Betrag vergrößert wird. Bei der linken Skizze wird diese Regel längs des Pfades ABDF angewandt; bei der mittleren Skizze wird sie dann längs des Pfades ACEF angewandt.

Es gibt jedoch noch den anderen Weg, den Fluß zu erhöhen: Wir können beliebige Pfade durch das Netzwerk betrachten, die Kanten enthalten können, die in die "falsche Richtung" zeigen (von der Senke zur Quelle längs des Pfades). Der Fluß längs eines solchen Pfades kann erhöht werden, indem der Fluß in von der Quelle zur Senke zeigenden Kanten *erhöht* und in von der Senke zur Quelle zeigenden Kanten um den gleichen Betrag *verringert wird*. In unserem Beispiel kann der Fluß durch das Netzwerk längs des Pfades ACDBEF um 3 erhöht werden, wie die dritte Skizze in der letzten Abbildung zeigt. Dies entspricht dem Hinzufügen von 3 Einheiten zum Fluß durch AC und CD und dem anschließenden Umleiten von 3 Einheiten am Schieber B von BD nach BE und EF. Der Fluß in DF verringert sich nicht, da 3 Einheiten, die zuvor von BD kamen, nun von CD kommen.

Algorithmus von Ford-Fulkerson, Komplexität

Eine Laufzeitschranke, die lediglich von der Größe des Graphen abhängt, erhält man, wenn man als zunehmenden Weg immer einen mit möglichst wenigen Pfeilen wählt. Bestimmt man solche kürzesten zunehmenden Wege für die einzelnen Flußvergrößerungensschritte, so vergrößert sich die Anzahl der Pfeile auf einem kürzesten Weg von q nach s nach höchstens $|E|$ Schleifendurchläufen wenigstens um 1. Damit ist die Anzahl der erforderlichen Iterationen beschränkt durch $(|V| - 1)\cdot|E|$. Weil man einen einzelnen zunehmenden Weg mittels Breitensuche in $O(|E|)$ Schritten finden kann, ergibt sich eine Laufzeit von insgesamt $O(|V|\cdot|E|^2)$ Schritten für das Berechnen eines maximalen Flusses.

Es gibt eine größere Zahl von Varianten und Verbesserungen zum Algorithmus von Ford-Fulkerson.