Teil 2:
Graphenalgorithmen

- Anwendungen
- Definitionen
- Datenstrukturen für Graphen
- Elementare Algorithmen
- Topologisches Sortieren
- Kürzeste Wege
- Minimal aufspannende Bäume
- Flüsse in Netzwerken
- Zusammenhangskomponenten

Problemstellung (1)

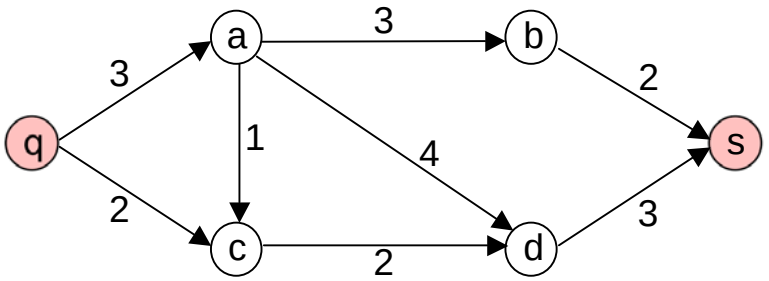
Netzwerk:

Ein Netzwerk ist ein gewichteter, gerichteter Graph mit zwei speziellen Knoten:

- Quelle (engl. source): Knoten ohne eingehende Kanten
- Senke (engl. sink): Knoten, ohne ausgehende Kanten.

Die Gewichte c(v,w) werden auch als Kapazitäten bezeichnet.

Beispiel:



Netzwerk mit Quelle q,
Senke s und Kapazitäten.

Anwendungen:

- Straßenverkehrsnetz:
  Kapazitäten geben maximal mögliche Verkehrsdichte je Straße an.
- Kanalisationsysytem:
  Kapazitäten geben maximale Wassermenge an, die je Zeiteinheit durch ein Rohr fließen kann.

Problemstellung (2)

Flüsse

Ein Fluß f in einem Netzwerk G ordnet jeder Kante (v,w) eine Zahl f(v,w) mit folgenden Eigenschaften zu:

(1) Kapazitätsbeschränkung: $0 \leq f(v,w) \leq c(v,w)$;

(2) für jeden Knoten (außer Quelle und Senke) gilt die Erhaltungseigenschaft:
die Summe der eingehenden Flüsse ist gleich der Summe der ausgehenden Flüsse.

Folgerung:

Die Summe der Flüsse, die aus der Quelle gehen, muss gleich der Summe der Flüsse sein, die an der Senke ankommen.

Wert eines Flusses:

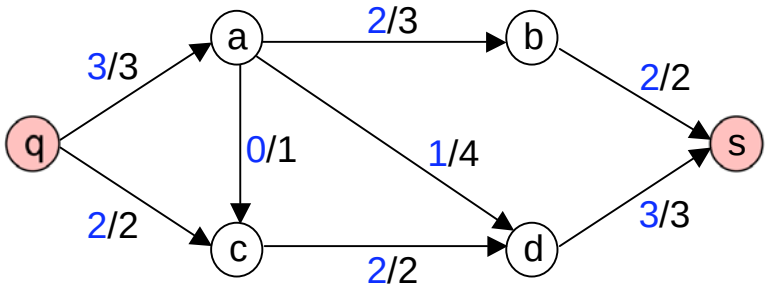
Der Wert eines Flusses ist gleich der Summe der Flüsse, die die Quelle verlassen.

Ziel: Bestimme den maximalen Fluss

Genauer: bestimme einen Fluss mit maximalem Wert.

Problemstellung (3)

Beispiel für Netzwerk mit maximalen Fluss:



Fluss / Kapazität

Maximaler Fluss hat den Wert 5.

Anwendungen:

- Welchen Verkehrsfluss verkraftet eine Stadt, für die ein Straßenverkehrsnetz gegeben ist.
- Welche Wassermenge lässt sich durch eine Kanalisation höchstens abtransportieren.

Algorithmus (1)

Grobstruktur:

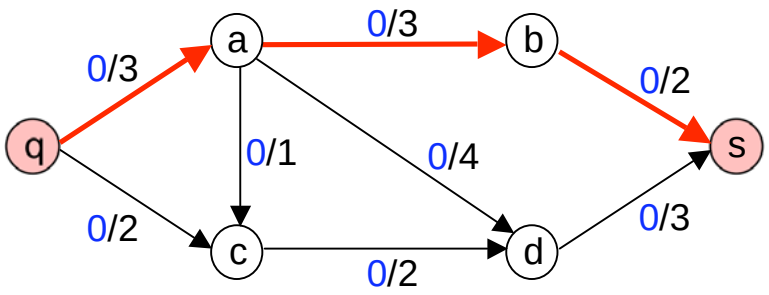
Starte mit Null-Fluss, d.h. f(v,w) = 0 für alle Kanten (v,w);

do {

(1) // Erweiterungsweg suchen:
Suche einen Weg von q nach s, bei dem jede Kante um einen Fluss Δf vergrößert werden kann;

(2) // Flusserweiterung:
vergrößere für jede Kante des Weges f um Δf;

} while (Fluss konnte erweitert werden);



Null-Fluss.

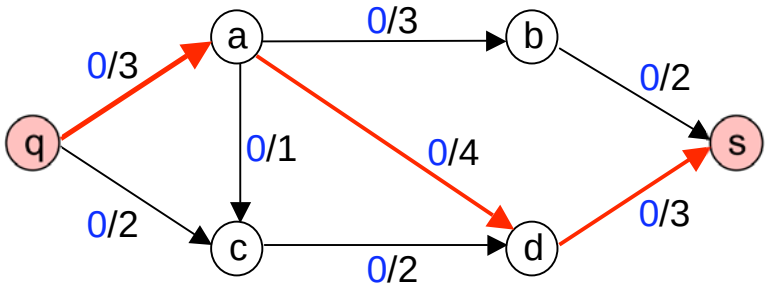
q, a, b, s ist ein möglicher Erweiterungsweg:
jede Kante kann um den Fluss $\Delta f = 2$ vergrößert werden.

Algorithmus (2)

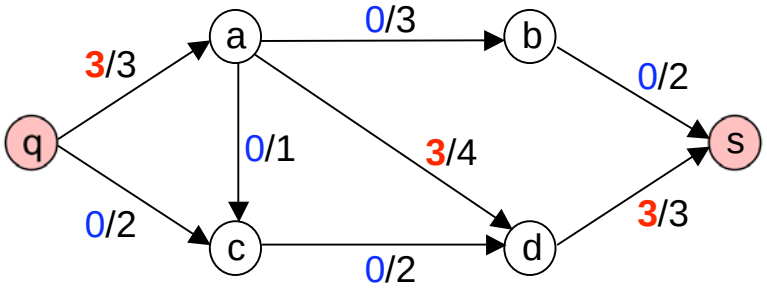
Problem:

Die Wahl eines Erweiterungsweges kann in eine Sackgasse führen (suboptimale Lösung).

Beispiel:



Erweiterungsweg,
bei dem jede Kante um $\Delta f = 3$ vergrößert werden kann.



Nach Flusserweiterung:

Fluss hat nun den Wert 3.

Problem: Es gibt nun keinen weiteren Erweiterungsweg, obwohl es eine Lösung mit einem Flusswert von 5 gibt (siehe S. 109)

Algorithmus (3)

Lösung:

Man erlaubt nach der Vergrößerung auch wieder eine Verkleinerung von Kantenflüssen.

Verwalte dazu außer Graph G mit den aktuellen Flüssen zusätzlich einen sogenannten **Residualgraphen** G_r (residual = als Rest zurückbleibend).

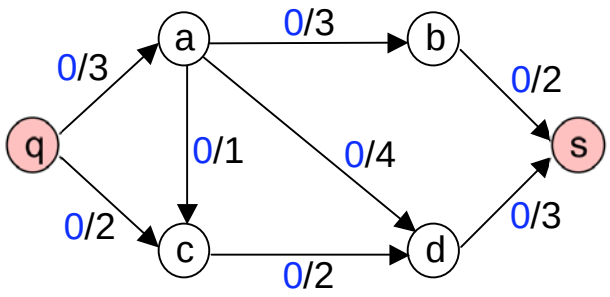
Im Residualgraph wird gespeichert, um welchen Wert der Fluss jeder Kante noch verändert werden darf (Residualfluss):

(1) f(v,w) > 0 darf verkleinert werden:
Hat eine Kante (v,w) den Fluss f(v,w) > 0, dann wird im Residualgraphen die Kante (w,v) mit dem Residualfluss f(v,w) abgespeichert.
Beachte: Kante in G_r verläuft in umgekehrter Richtung.
(Flusserhöhung in umgekehrter Richtung verkleinert den Fluss.)

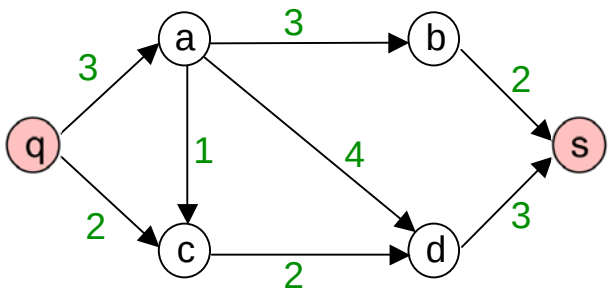
(2) f(v,w) < c(v,w) darf vergrößert werden:
Hat eine Kante (v,w) den Fluss f(v,w) < c(v,w), dann wird im Residualgraphen die Kante (v,w) mit Residualfluss c(v,w) - f(v,w) abgespeichert.

Algorithmus (3)

Beispiel: Residualgraph für Nullfluss



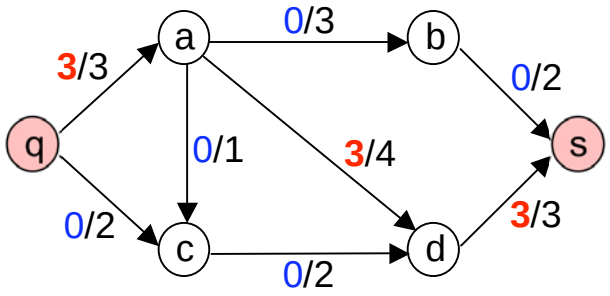
Graph G mit Nullfluss.



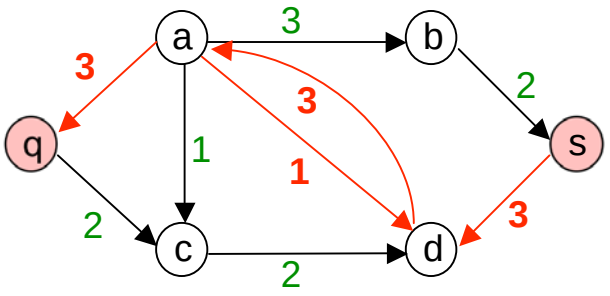
Residualgraph G_r mit Residualflüssen

Beispiel: Residualgraph nach Flusserweiterung

q, a, d, s wird als Erweiterungsweg gewählt. Fluss lässt sich um $\Delta f = 3$ vergrößern.



Graph G mit Flusswert 3.



Residualgraph G_r mit Residualflüssen

Fluss von q nach a darf um bis zu 3 verringert werden.

Fluss von a nach d darf um bis zu 1 erhöht und bis zu 3 verringert werden.

Fluss von d nach s darf um bis zu 3 verringert werden.

Algorithmus (4)

Endgültige Formulierung:

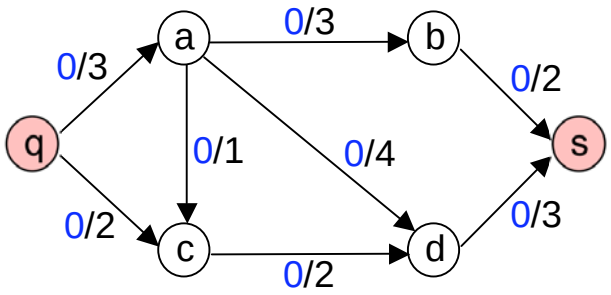
```
Initialisiere Graph G mit Null-Fluss, d.h. f(v,w) = 0 für alle Kanten (v,w);
Initialisiere Residualgraph G_r;

do
{
(1)   // Erweiterungsweg suchen:
      Suche im Residualgraphen G_r einen Weg p von q nach s;
      Δf = Minimum aller Residual-Flüsse im Weg p;

(2)   // Fluss vergrößern:
      vergrößere für jede Kante (v,w) im Weg p den Fluss um Δf
      (beachte dabei, dass umgekehrte Residualflüsse den Fluss verkleinern);
      führe entsprechende Änderungen im Residualgraphen durch;

} while (Fluss konnte vergrößert werden);
```

2 Ansätze, um einen Erweiterungsweg von q nach s im Residualgraphen zu suchen:

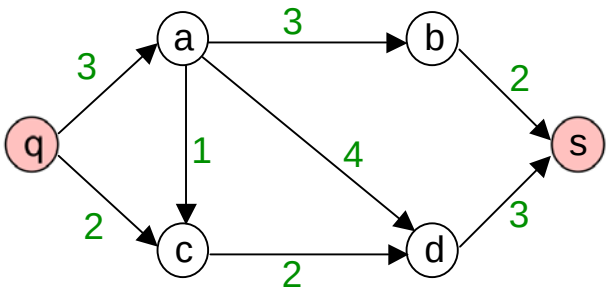
- suche den Weg mit dem größten Δf
  (wie bei kürzeste Wege in Distanzgraphen; siehe Algorithmus von Dijkstra)
- suche Weg mit kleinster Kantenzahl
  (wie kürzeste Wege in ungewichteten Graphen durch erweiterte Breitensuche)

Beispiel (1)

Starte mit Nullfluss



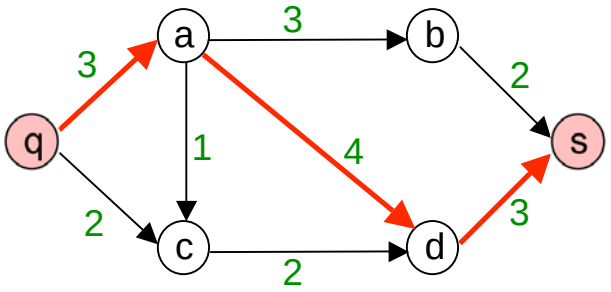
Graph G mit Nullfluss



Residualgraph G_r mit Residualflüssen

Beispiel (2)

Suche Weg mit größten Δf im Residualgraphen:

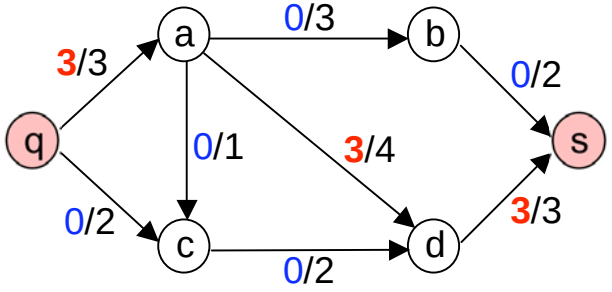


Residualgraph G_r

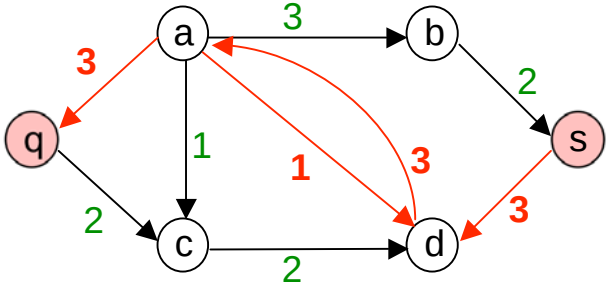
q, a, d, s ist Weg mit größtem Δf.

Δf = Minimum der Residual-Flüsse im Weg = 3.

Flusserweiterung um Δf:



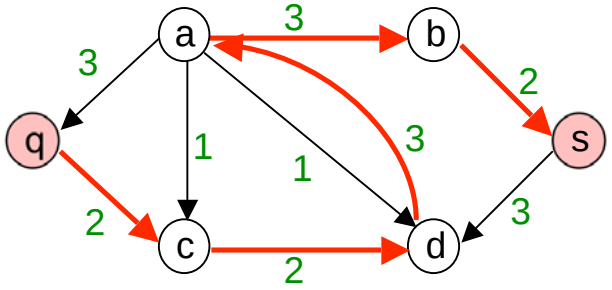
Graph G mit Flusswert 3



Residualgraph G_r

Beispiel (3)

Suche Weg mit größten Δf im Residualgraphen:

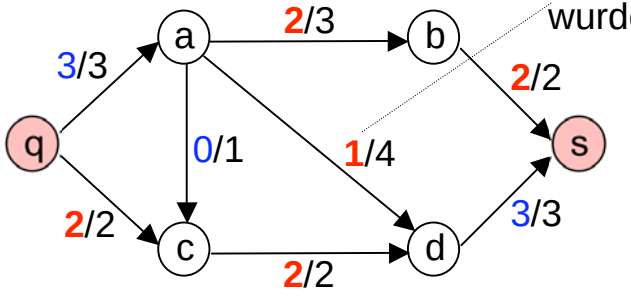


Residualgraph G_r

q, c, d, a, b, s ist Weg mit größtem Δf.

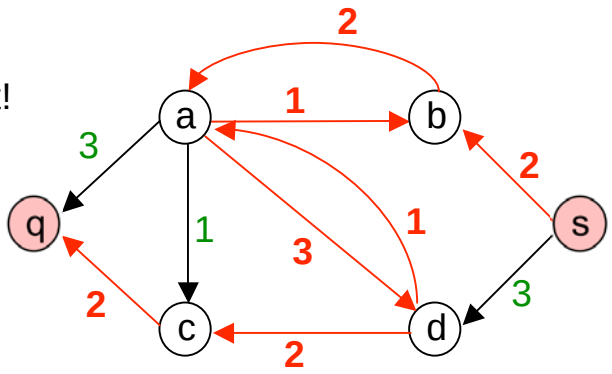
Δf = Minimum der Residual-Flüsse im Weg = 2.

Flusserweiterung um Δf:



Beachte: Fluss wurde verkleinert!

Graph G mit Flusswert 5



Residualgraph G_r

Ende:

Es gibt im Residualgraphen keine Wege mehr von q nach s. Maximaler Fluss gefunden!

Analyse und Bemerkungen

Analyse:

$T = O(|E|^2 \log|V|)$

(Beweis siehe [Turau 2004]).

Bemerkungen:

- Der hier beschriebene Algorithmus geht zurück auf Ford und Fulkerson (1956). Analyse von Karp und Edmonds (1972).
- Man beachte, dass bei einem dichten Graphen ($|E| = O(|V|^2)$) die Laufzeit anwächst auf $T = O(|V|^4 \log|V|)$
- Es gibt inzwischen wesentlich schnellere Algorithmen:
  Preflow-Push-Algorithmus von Goldberg, 1985: $T = O(|V|^3)$.
  Mittels spezieller Datenstrukturen erreichte Goldberg 1988 sogar:
  $T = O(|E|\ |V| \log(|V|^2/|E|)$.

Andere Flussprobleme:

- Kostenminimale Flüsse

Teil 2:
Graphenalgorithmen

- Anwendungen
- Definitionen
- Datenstrukturen für Graphen
- Elementare Algorithmen
- Topologisches Sortieren
- Kürzeste Wege
- Minimal aufspannende Bäume
- Flüsse in Netzwerken
- Zusammenhangskomponenten

Zusammenhangskomponenten (1)

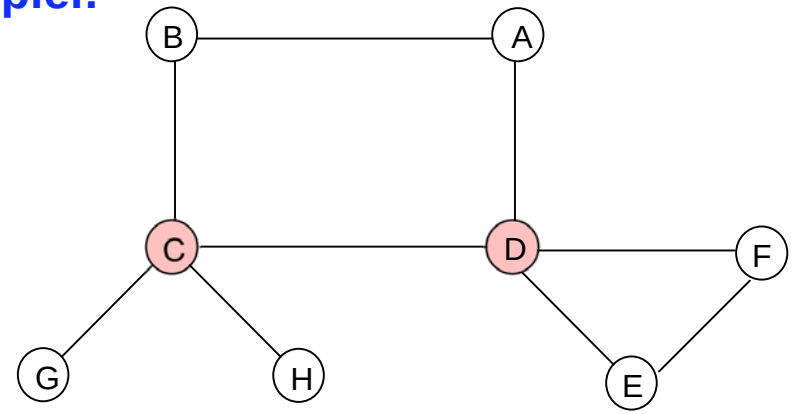
Definitionen:

- Ein ungerichteter Graph heißt zusammenhängend (engl. connected), falls es zu jedem Knoten einen Weg zu jeden anderen Knoten gibt.
- Eine Zusammenhangskomponente eines ungerichteten Graphen G ist ein maximal zusammenhängender Teilgraph.
- Ein Knoten heißt Artikulationspunkt (Artikulation med. = Gelenk; engl. articulation point), wenn sein Wegfall die Anzahl der Zusammenhangskomponenten erhöht.
- Ein Graph heißt zweifach zusammenhängend (engl. biconnected), falls er keinen Artikulationspunkt besitzt.

Problem:

Bestimme in einem ungerichteten Graphen alle Artikulationspunkte.

Beispiel:



Ein ungerichteter Graph G mit den Artikulationspunkten C und D.

G \ {C} zerfällt in 3 Zusammenhangs-komponenten und

G \ {D} zerfällt in 2 Zusammenhangs-komponenten.

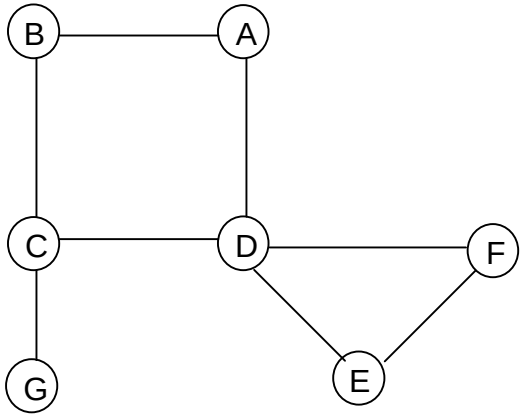
Zusammenhangskomponenten (2)

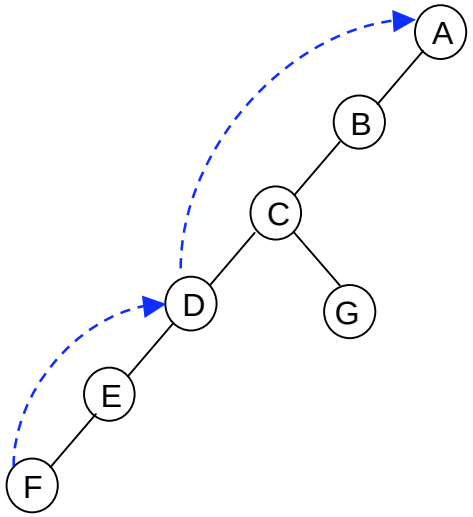
Tiefensuchbaum mit Rückwärtskanten (TSB)

Stellt man die Tiefensuche graphisch dar, ergibt sich ein Baum, der Tiefensuchbaum.

Rückwartskanten sind Kanten, die einen Knoten mit einem bereits früher besuchten Knoten verbinden (jedoch nicht den unmittelbar davor besuchten Knoten)

Beispiel:





Tiefensuchbaum, der mit Knoten A beginnt.
Rückwärtskanten sind gestrichelt.

Die schwarzen Kanten werden auch Vorwärtskanten genannt.

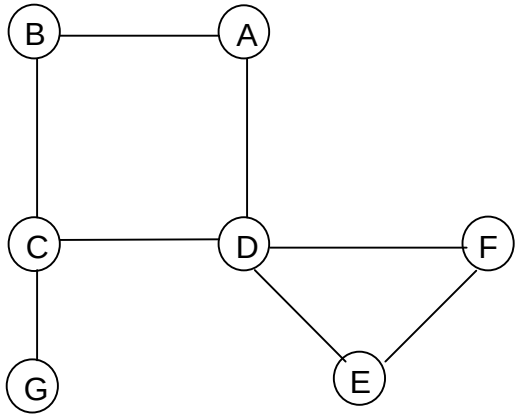
Zusammenhangskomponenten (3)

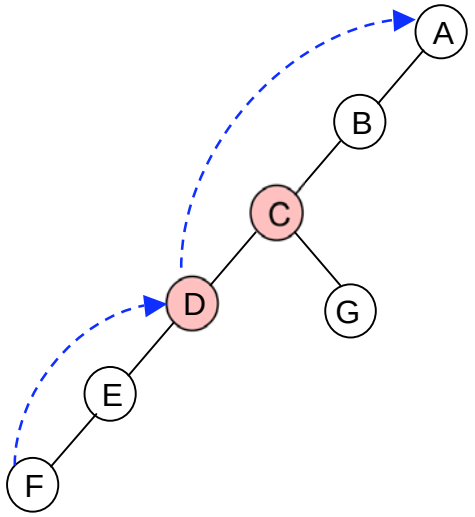
Artikulationspunkte lassen sich in einem TSB wie folgt erkennen:

(1) Die Wurzel ist ein Artikulationspunkt, genau dann wenn sie mehr als einen Nachfolger hat. (Betrachte z.B. für den unten abgebildeten Graphen ein TSB, der mit Knoten C beginnt.)

(2) Ein Knoten v ist ein Artikulationspunkt, falls v im TSB einen Nachfolger hat, von dem es keinen Weg über eine Rückwartskante zu einem Vorgänger von v gibt.

Beispiel:





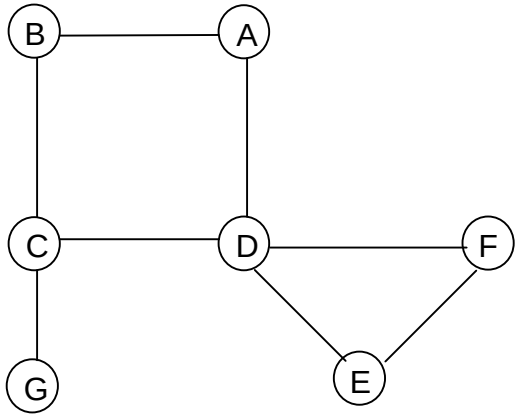
C und D sind wegen Regel (2) Artikulationspunkte.

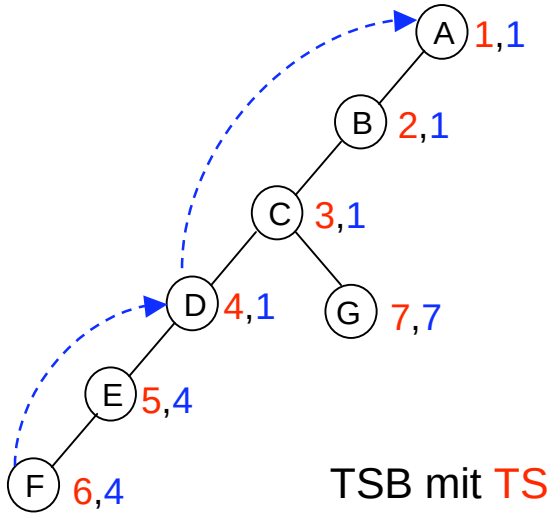
Zusammenhangskomponenten (4)

Zwei Nummerierungen,
mit deren Hilfe Artikulationspunkte bestimmt werden:

- TSNr[v]: gibt für jeden Knoten v, den Besuchszeitpunkt bei der Tiefensuche an.
- MinNr[v]: die kleinste TSNr eines Knoten w, so dass w von v erreicht wird, indem keine, eine oder mehrere Vorwärtskanten und dann eventuell eine Rückwärtskante genommen werden:

Beispiel:





TSB mit TSNr und MinNr

Regel (2) lässt sich damit wie folgt umsetzen:

Knoten v (außer Wurzel) ist ein Artikulationspunkt gdw.
v ein Kind w hat mit MinNr[w] ≥ TSNr[v] .

Zusammenhangskomponenten (5)

Berechnung von MinNr:

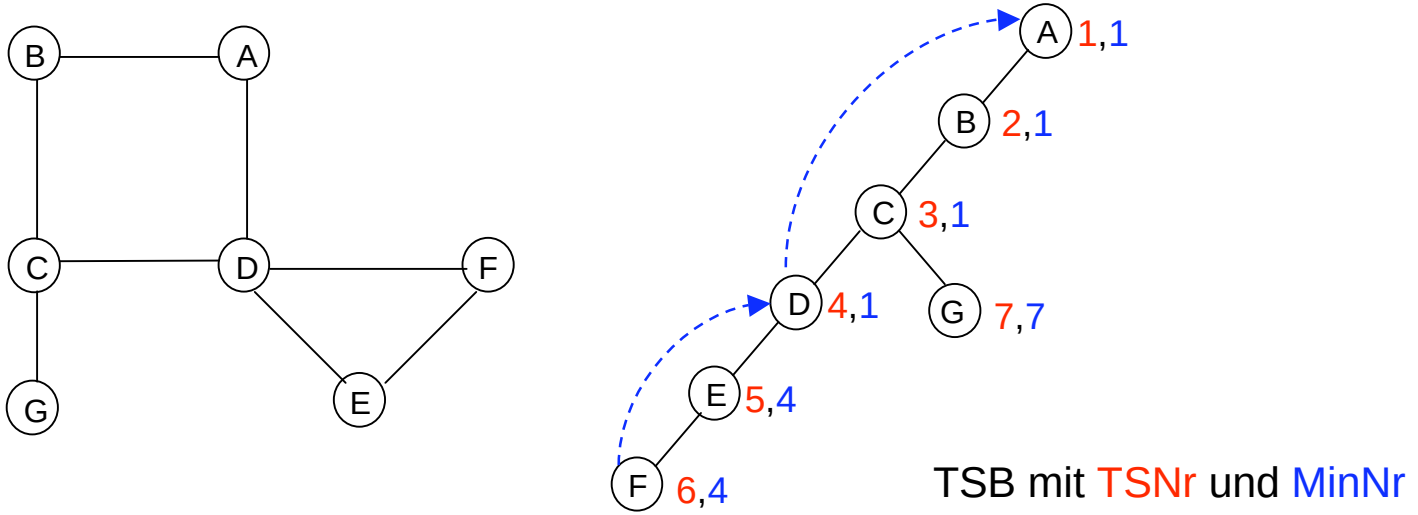
MinNr[v] = Minimum von

(M1) TSNr[v],

(M2) kleinster TSNr[w] für alle Rückwärtskanten (v,w)

(M3) kleinster MinNr[w] für alle Vorwärtskanten (v,w)

Beispiel:



Zusammenhangskomponenten (6)

```
bool visited[n] = false;
int TSNr[n];
int counter = 1;
int MinNr[n];
int p[n];

void findArtPoint(Vertex v)
{
      visited[v] = true;
      TSNr[v] = counter++;
      MinNr[v] = TSNr[v];    // Regel (M1)

      for (jeden Nachbar w von v )
            if (! visited[w]) {
                  p[w] = v;
                  findArtPoint(w);
                  if (MinNr[w] >= TSNR[v])
                        cout << v << “ist ein Artikulations-Punkt” << endl;
                  MinNr[v] = Minimum(MinNr[v], MinNr[w]);   // Regel (M3)
            }
            else if (p[v] != w)   // (v,w) ist Rückwärtskante
                  MinNr[v] = Minimum(MinNr[v], TSNr[w]);    // Regel (M2)
}

Wähle beliebigen Knoten v;
findArtPoint(v);
```

Globale Variablen:

counter dient zum Hochzählen der Besuchszeitpunkte.

p[v] speichert den im TSB zu v gehörenden Elternknoten ab.

Die rekursive Funktion findArtPoint ist ein erweiterter Tiefensuch-Algorithmus.

Die Prüfung, ob die Wurzel ein Artikulationspunkt ist (Regel (1)), wurde einfachheitshalber weggelassen.