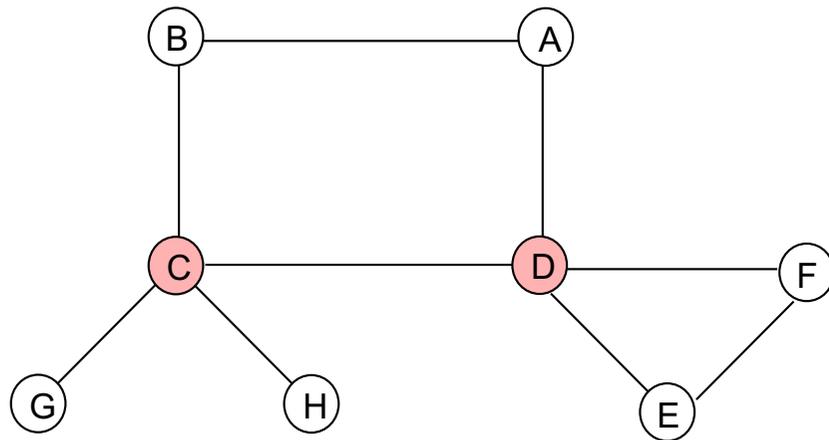


12. Zweifach zusammenhängende Graphen

- Artikulationspunkte, zweifach zusammenhängende Graphen
- Tiefensuchwald mit Rückwärtskanten
- Algorithmus zur Berechnung der Artikulationspunkte

Artikulationspunkt

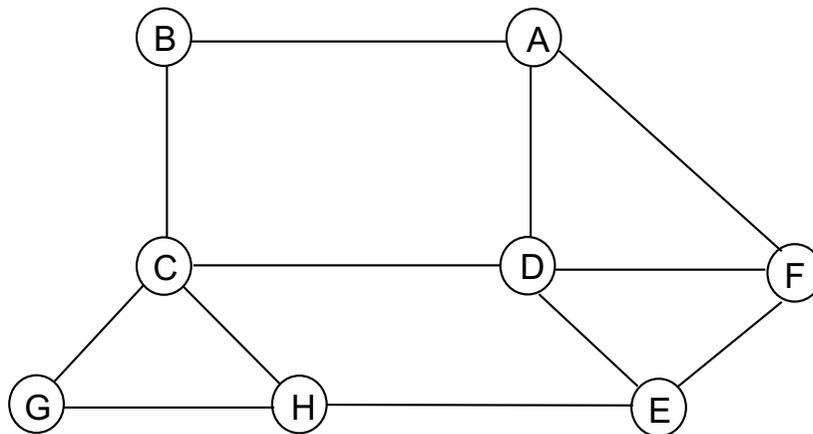
- Ein ungerichteter Graph heißt **zusammenhängend** (engl. connected), falls es für jeden Knoten einen Weg zu jedem anderen Knoten gibt.
- Eine **Zusammenhangskomponente** eines ungerichteten Graphen G ist ein maximal zusammenhängender Teilgraph.
- Ein Knoten heißt **Artikulationspunkt** (Artikulation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



- Ein ungerichteter und zusammenhängender Graph G mit den Artikulationspunkten C und D.
- $G \setminus \{C\}$ zerfällt in 3 Zusammenhangskomponenten
- $G \setminus \{D\}$ zerfällt in 2 Zusammenhangskomponenten.

Artikulationspunkte und zweifacher Zusammenhang

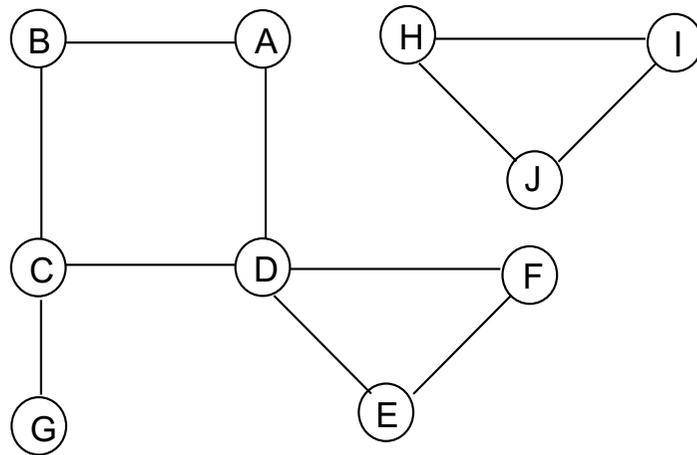
- Ein Graph heißt **zweifach zusammenhängend** (engl. biconnected), falls er zusammenhängend ist und keinen Artikulationspunkt besitzt.
- **Ziel:** Bestimme in einem ungerichteten Graphen alle Artikulationspunkte.
- Damit ist auch das Problem des zweifachen Zusammenhangs gelöst.



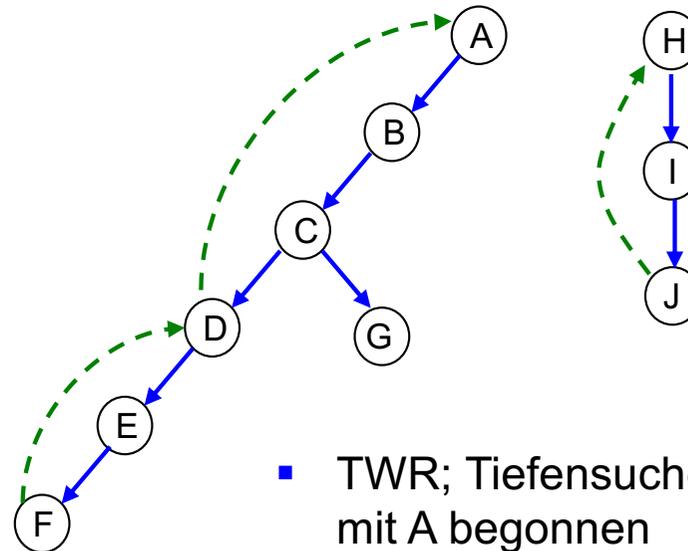
- Graph G ist zusammenhängend und hat keine Artikulationspunkte
- G ist daher zweifach zusammenhängend

Tiefensuchwald mit Rückwärtskanten (TWR)

- Tiefensuche für einen Graph G ergibt einen **Tiefensuchwald**.
- Der Tiefensuchwald besteht aus **Vorwärtskanten**.
- Der Tiefensuchwald wird ergänzt um **Rückwärtskanten**.
Rückwärtskanten sind Kanten, die einen Knoten mit einem bereits früher besuchten Knoten verbinden jedoch nicht den unmittelbar davor besuchten Knoten.
- Beachte: Anzahl der Kanten in G und Anzahl der Vorwärts- und Rückwärtskanten in TWR müssen gleich sein.



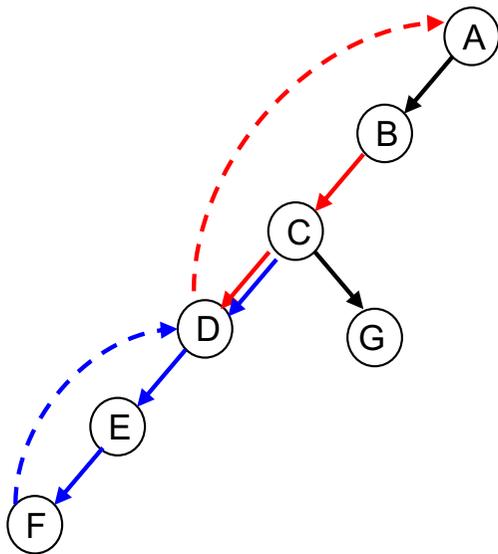
Graph G



- TWR; Tiefensuche wurde mit A begonnen
- **Vorwärtskanten** und **Rückwärtskanten**

Vorfahre und Rückwärtsweg

- Knoten v ist **Vorfahre** von w , falls es im TWR einen Weg von v nach w gibt, der nur aus Vorwärtskanten besteht.
- Ein **Rückwärtsweg** ist ein Weg in einem TWR mit einer beliebig langen (evtl. leeren) Folge von Vorwärtskanten und dann genau einer Rückwärtskante.



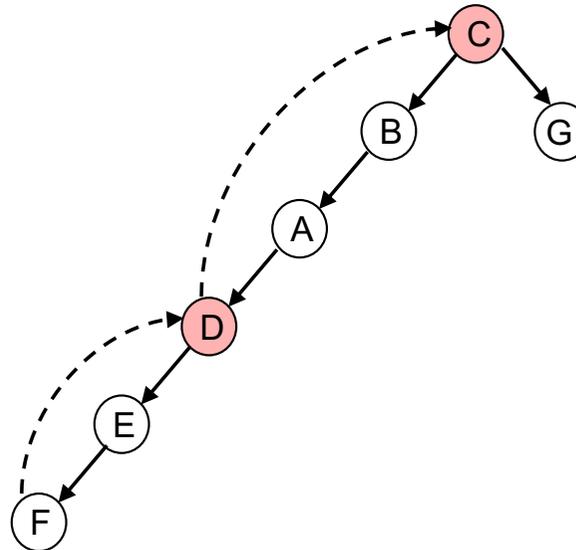
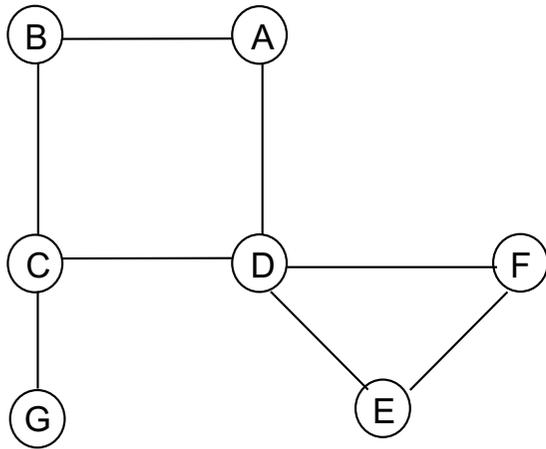
Beispiele:

- C ist Vorfahre von D, E, F und G
- **B – C – D – A ist ein Rückwärtsweg**
- **C – D – E – F – D ist ein Rückwärtsweg**
- E – F – D – A ist kein Rückwärtsweg

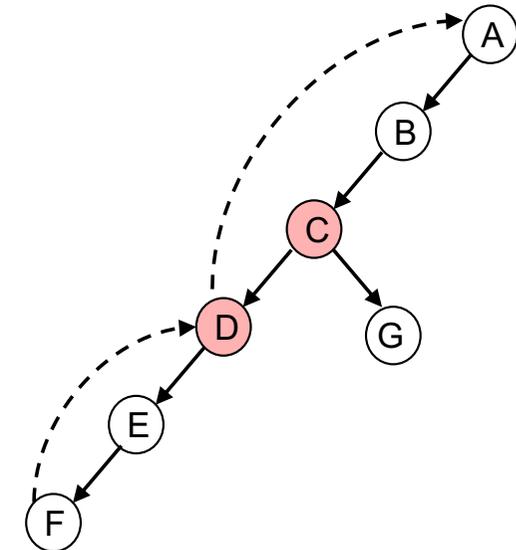
Bestimmung von Artikulationspunkten

(W) Die **Wurzel** ist ein Artikulationspunkt, wenn sie mehr als ein Kind hat.

(NW) Ein Knoten $v \neq \text{Wurzel}$ ist ein Artikulationspunkt, falls v im TWR ein Kind w hat, so dass es von w keinen Rückwärtsweg zu einem Vorfahren von v gibt.



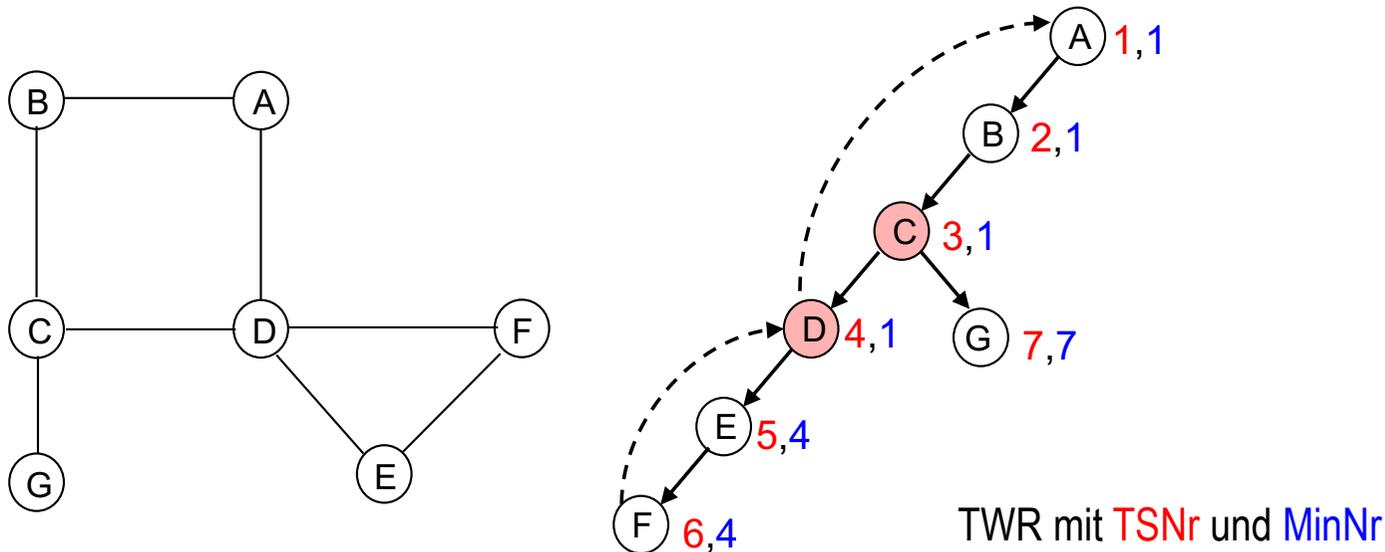
- TWR, der mit Knoten C beginnt
- C ist Artikulationspunkt wegen (W)
- D ist Artikulationspunkt wegen (NW)



- TWR, der mit Knoten A beginnt
- C und D sind Artikulationspunkte wegen Regel (NW)

Nummerierungen: TSNr und MinNr

- **TSNr[v]:** Besuchszeitpunkt für Knoten v bei der rekursiven Tiefensuche
- **MinNr[v]:** Minimum { TSNr[w] / w = v oder w ist von v über einen Rückwärtsweg erreichbar }
Wie weit kommt man von v über einen Rückwärtsweg zurück?

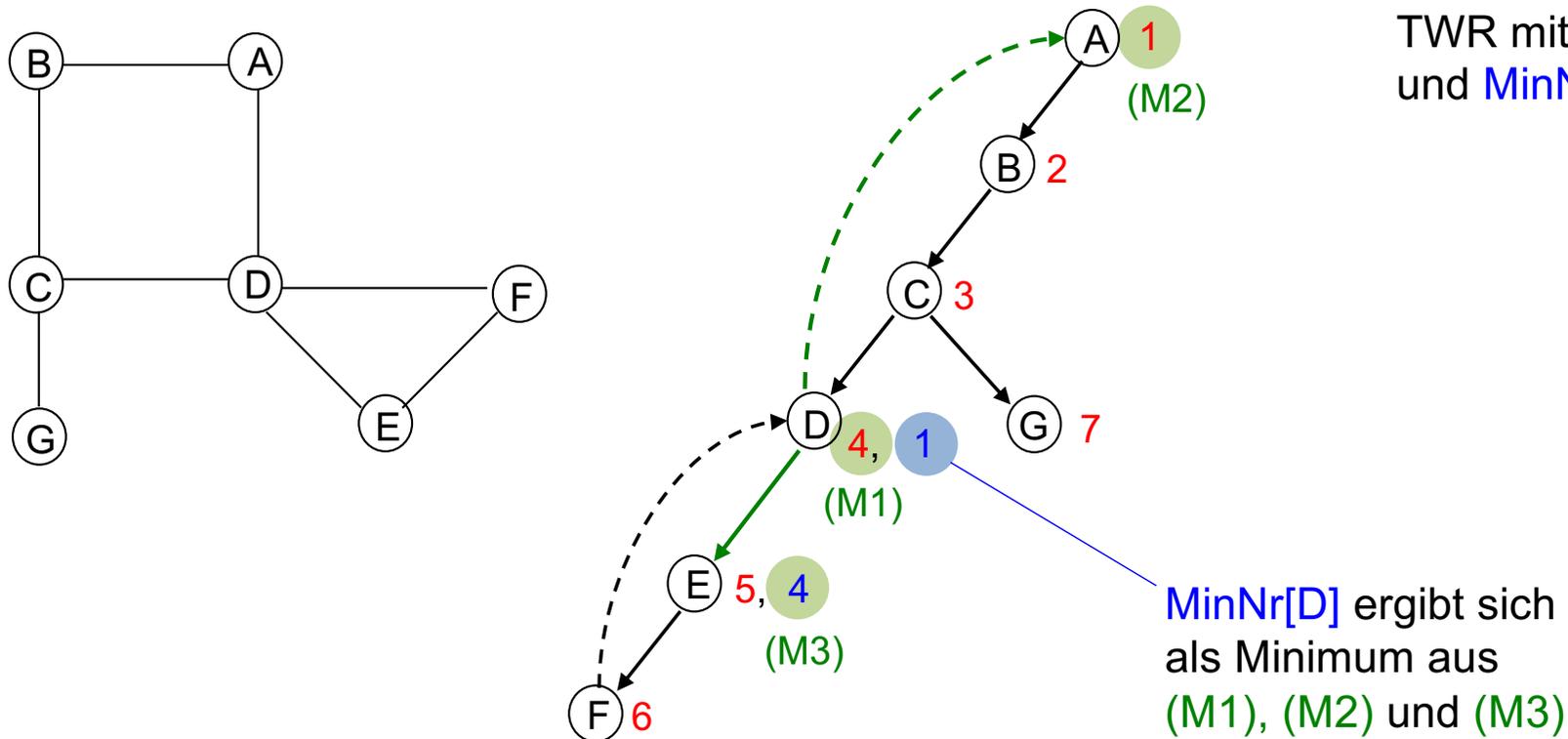


Damit lässt sich Regel (NW) neu formulieren:

(NW) Knoten $v \neq$ Wurzel ist genau dann ein Artikulationspunkt, wenn v ein Kind w hat mit $\text{MinNr}[w] \geq \text{TSNr}[v]$.

Berechnung von MinNr

- MinNr[v] = Minimum von
 - (M1) TSNr[v],
 - (M2) kleinster Wert von TSNr[w] für alle Rückwärtskanten (v,w)
 - (M3) kleinster Wert von MinNr[w] für alle Vorwärtskanten (v,w)



Algorithmus zur Bestimmung von Artikulationspunkten (1)

```
Set<Vertex> visited;  
int counter;  
int[ ] TSNr;  
int[ ] MinNr;  
Vertex[ ] p;  
  
public void findArtPoint(Graph g) {  
    visited = ∅;  
    counter = 1;  
  
    for (jeden Knoten v)  
        if (! visited.contains(v) )  
            findArtPoint(g, v);  
}  
  
private void findArtPoint(Graph g, Vertex v) {  
    // s. nächste Seite  
}
```

Globale Variablen:

- counter dient zum Hochzählen der Besuchszeitpunkte.
- TSNr und MinNr wie soeben beschrieben.
- p[v] speichert den im TWR zu v gehörenden Elternknoten ab.

- findArtPoint(g) gibt alle Artikulationspunkte des ungerichteten Graphen g aus.
- findArtPoint(g,v) startet eine rekursive Tiefensuche bei Knoten v.

- Beachte:
Die Prüfung, ob eine Wurzel ein Artikulationspunkt ist (Regel (W)), wurde einfachheitshalber weggelassen, lässt sich aber einfach ergänzen.

Algorithmus zur Bestimmung von Artikulationspunkten (2)

```
Set<Vertex> visited;
int counter;
int[ ] TSNr[n];
int[ ] MinNr;
int[ ] p;

// ...

private void findArtPoint(Graph g, Vertex v) {
    visited.add(v);
    TSNr[v] = counter++;
    MinNr[v] = TSNr[v]; // (M1)
    for (jeden Nachbarn w von v) {
        if (! visited.contains(w)) {
            p[w] = v;
            findArtPoint(g, w);
            if (MinNr[w] >= TSNr[v]) println(v + "ist ein Artikulations-Punkt");
            MinNr[v] = Minimum(MinNr[v], MinNr[w]); // (M3)
        }
        else if (p[v] != w) // (v,w) ist Rückwärtskante
            MinNr[v] = Minimum(MinNr[v], TSNr[w]); // (M2)
    }
}
```

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

▪ Mit p[v] kann geprüft werden, ob (v,w) tatsächlich eine Rückwärtskante ist.