Algorithmen und Datenstrukturen

Teil 5:
Algorithmen auf Graphen (I) – Transitiver Abschluss, Aufspannende Bäume und Suchverfahren

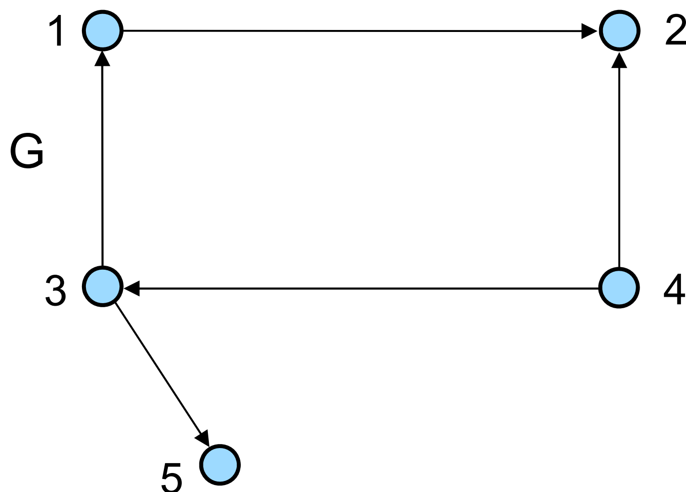
DHBW Stuttgart Campus Horb
Fakultät Technik
Studiengang Informatik
Dozent: Olaf Herden
Stand: 04/2021

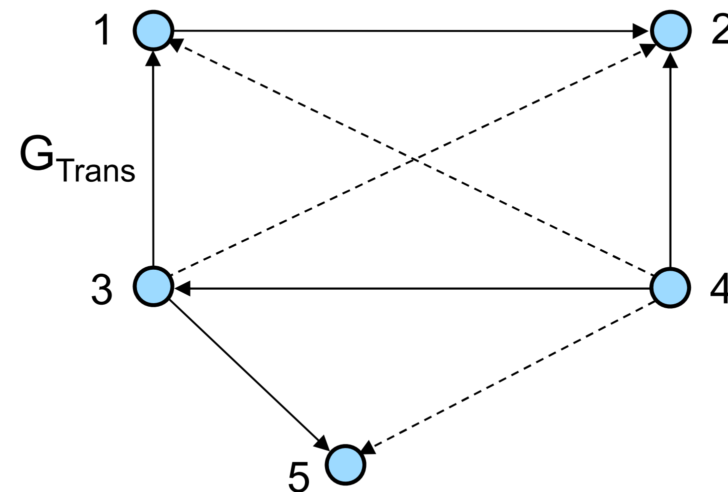
Gliederung

- Transitiver Abschluss
- Minimal aufspannende Bäume
- Suchverfahren in Graphen
- Breitensuche
- Tiefensuche

Transitiver Abschluss eines Graphen

- Transitiver Abschluss eines gerichteten Graphen: Existieren Kanten (v_1,v_2) und (v_2,v_3) ⇒ Füge Kante (v_1,v_3) hinzu
- Beispiel: G_{Trans} ist transitiver Abschluss von G





- Anwendungsbeispiel:
  - Vererbungshierarchie in Java:
    - Ist Klasse Unterklasse einer anderen?
    - Transitiver Abschluss beantwortet Frage durch die Existenz eines Vorgängers unmittelbar

Berechnung transitiver Abschluss

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

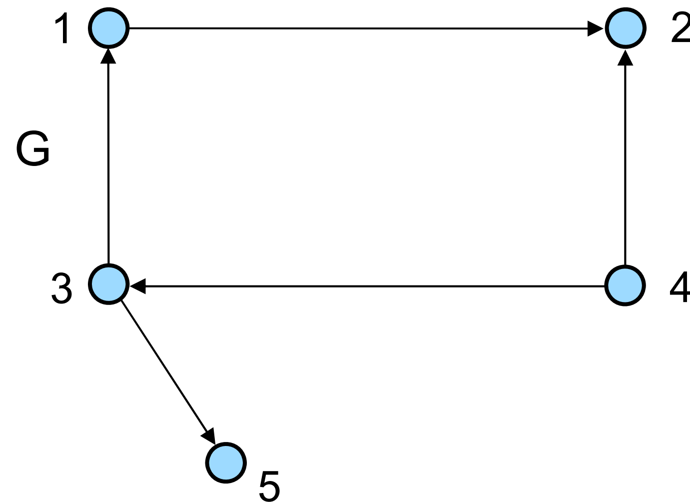
- Systematische Berechnung:
  - Verfahren 1: Ermittlung der Erreichbarkeitsmatrix durch Adjazenzmatrizenmultiplikation (Adjazenzmatrizenmultiplikationsverfahren)
  - Verfahren 2: Sukzessives Ermitteln (Warshall-Algorithmus)

- Satz als Basis für Verfahren 1:
  - Sei G gerichteter Graph mit Adjazenzmatrix A und A^s die s-te Potenz von A
  - Dann gilt: Wert an Stelle (i,j) von A^s gibt Anzahl verschiedener Kantenzüge der Länge s vom Startknoten i zum Zielknoten j an.
  - Beweis: Siehe z.B. [Tura04].

Adjazenzmatrizenmultiplikationsverfahren (I)

- Daraus folgt Algorithmus:
  - G habe n Knoten
  - Gibt es Weg zwischen v_i und v_j $\Rightarrow$ Zwischen v_i und v_j gibt es auch Weg der maximal Länge n-1
  - Mit anderen Worten: Ist der Knoten i vom Knoten j aus erreichbar, dann gibt es eine Zahl s mit $1 \leq s \leq n-1$, so dass an der Stelle (i,j) in A^s ein Wert ungleich 0 steht
  - Damit kann aus der Matrix $E = \sum_{s=1}^{n-1} A^S$ die Adjazenzmatrix des transitiven Abschluss gebildet werden: $e_{ij} = \begin{cases} 1 & falls\ s_{ij} \neq 0 \\ 0 & sonst \end{cases}$
  - Anm.: E steht für Erreichbarkeitsmatrix

Beispiel (I)



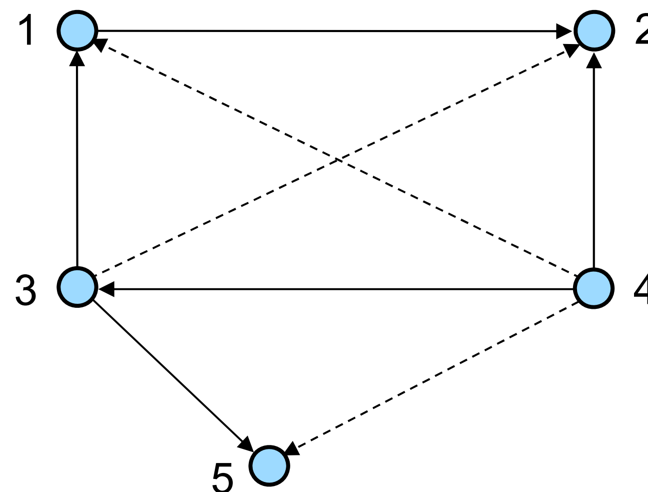
$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad A^2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad A^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Beispiel (II)

$$A^4 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

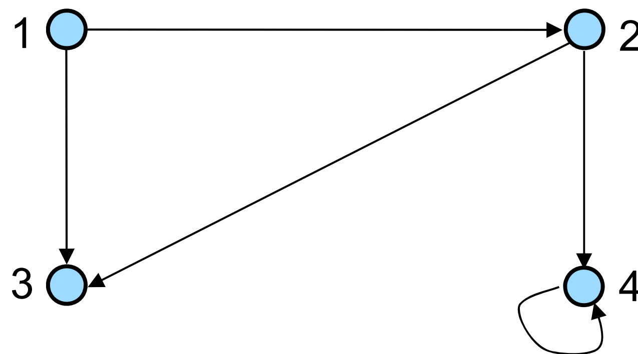
- Damit ergibt sich folgende Erreichbarkeitsmatrix und damit der transitive Abschluss:

$$E = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



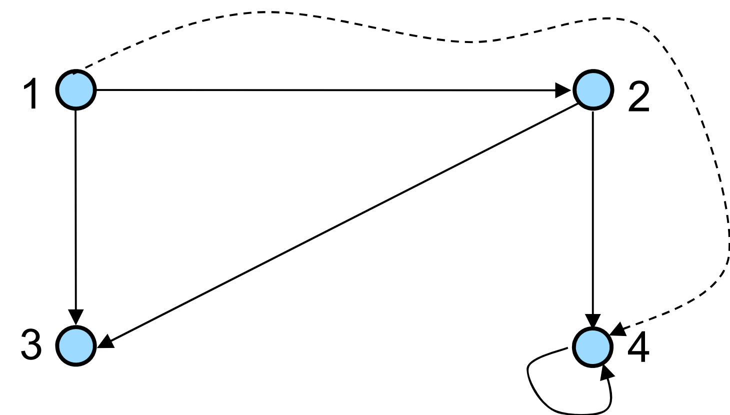
Anmerkungen

- Verfahren funktioniert auch für Graphen mit Schlingen
- Allerdings: Aussage, dass A^s die Anzahl der Kantenzüge der Länge s enthält, stimmt nicht mehr
- Beispiel:



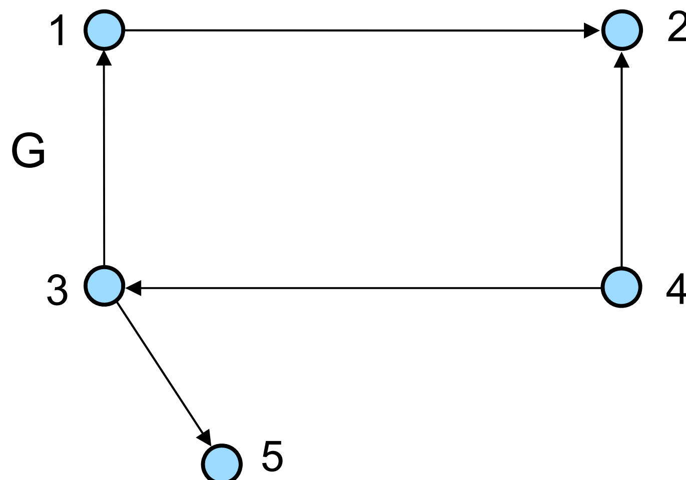
$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A^2 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A^3 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad E = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Warshall-Algorithmus

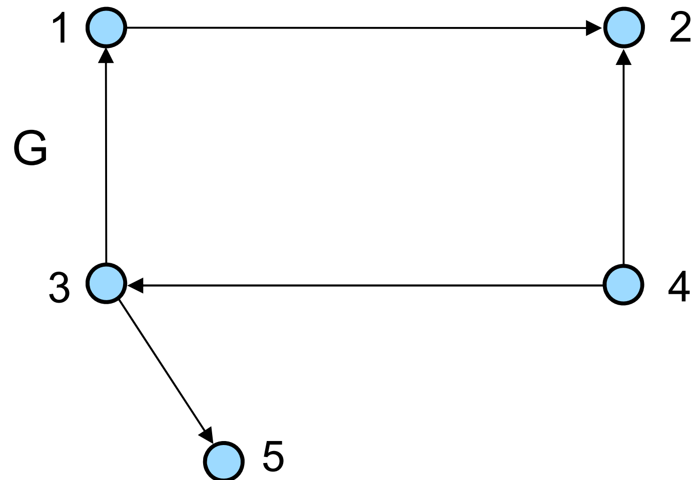
- Prinzip:
  - Sei G gerichteter Graph
  - Adjazenzmatrix A(G) wird in n Schritten in Adjazenzmatrix des transitiven Abschlusses überführt
  - Aus dem i-tem Schritt resultierender Graph wird mit G_i bezeichnet ($G=G_0$)
  - Übergang von G_k nach G_{k+1}:
    - Füge jedem Knotenpaar i, j Kante von i nach j hinzu, falls in G_k Kanten von i nach k+1 und von k+1 nach j existieren
- Beispiel:



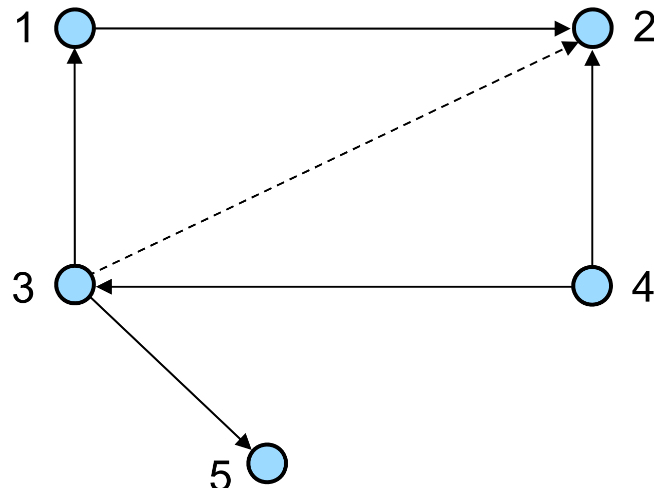
  - Knotenreihenfolge für die Abarbeitung: 1, 2, 3, 4, 5

Beispiel Warshall-Algorithmus (I)

- G_0 ist der Ausgangsgraph:

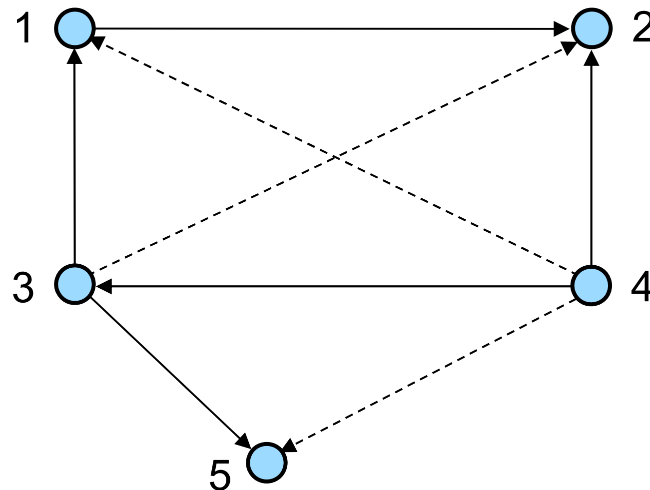


- $G_1 = G_0 + (3,2)$, da Kante (3,1) und (1,2) existiert (k=1)



Beispiel Warshall-Algorithmus (II)

- $G_2 = G_1$, da keine Kante die Bedingung erfüllt
- $G_3 = G_2$ + (4,5), da Kante (4,3) und (3,5) existiert (k=3)
  + (4,1), da Kante (4,3) und (3,1) existiert (k=3)
  [+ (4,2), da Kante (4,3) und (3,2) existiert (k=3)]



Wenn diese Kante nicht schon da wäre, würde sie jetzt eingefügt werden

- $G_4 = G_3$, da keine Kante die Bedingung erfüllt
- $G_5 = G_4$, da keine Kante die Bedingung erfüllt

Anmerkungen Warshall-Algorithmus

- In einem Schritt können u.U. auch mehrere Kanten eingefügt werden:
  - Hat der aktuell betrachtete Knoten n eingehende und m ausgehende Kanten, können bis zu n*m neue Kanten eingefügt werden

- Abarbeitungsreihenfolge der Knoten ist egal:
  - Stets wird das gleiche Resultat erreicht
  - Unterschiedlich ist aber die Reihenfolge, in der die neuen Kanten hinzugefügt werden

Komplexität des transitiven Abschluss

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Adjazenzmatrizenmultiplikationsverfahren:
  - Bestimme Potenzen A^s der Adjazenzmatrix A für s = 1, ..., n-1 und addiere diese:
    - Einfache Verfahren zur Multiplikation zweier nxn-Matrizen: $O(n^3)$
    - Addieren: $O(n^2)$
    - Aktionen hintereinander ⇒ Gesamtaufwand $O(n^3)$
- Warshall-Algorithmus:
  - Drei ineinander geschachtelte for-Schleifen ⇒ $O(n^3)$
- Fazit: Im worst case unterscheiden sich beiden Algorithmen nicht!
- Speicherplatzbedarf:
  - Bei Adjazentmatrizenmultiplikationsverfahren höher (zu jedem Zeitpunkt drei Matrizen)
  - Warshall-Algorithmus eine Matrix

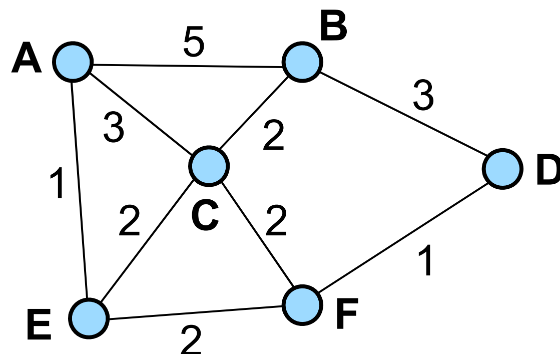
Übersicht

- Transitiver Abschluss
- Minimal aufspannende Bäume
- Suchverfahren in Graphen
- Breitensuche
- Tiefensuche

Motivation/Beispiel

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

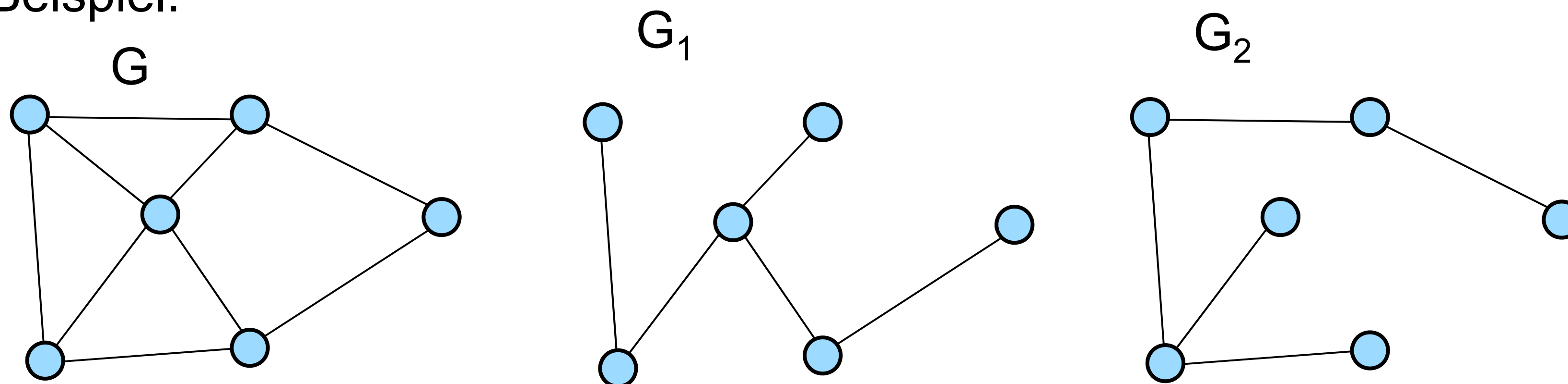
- Planung eines Kommunikationsnetzes:
  - Zwischen n Orten ist Kommunikationsnetz zu planen
  - Je zwei verschiedene Orte sollen miteinander verbunden werden (entweder direkt oder indirekt über andere Orte)
  - Verzweigungspunkte sind nur in Orten erlaubt
  - Kosten für einzelne Direktverbindungen sind bekannt
  - Gesucht: Kommunikationsnetz mit minimalen Kosten
  - Beispiel:



- Wie sieht das zugehörige Kommunikationsnetz (Graph) mit minimalen Kosten aus?

Definition (I): Aufspannender Baum

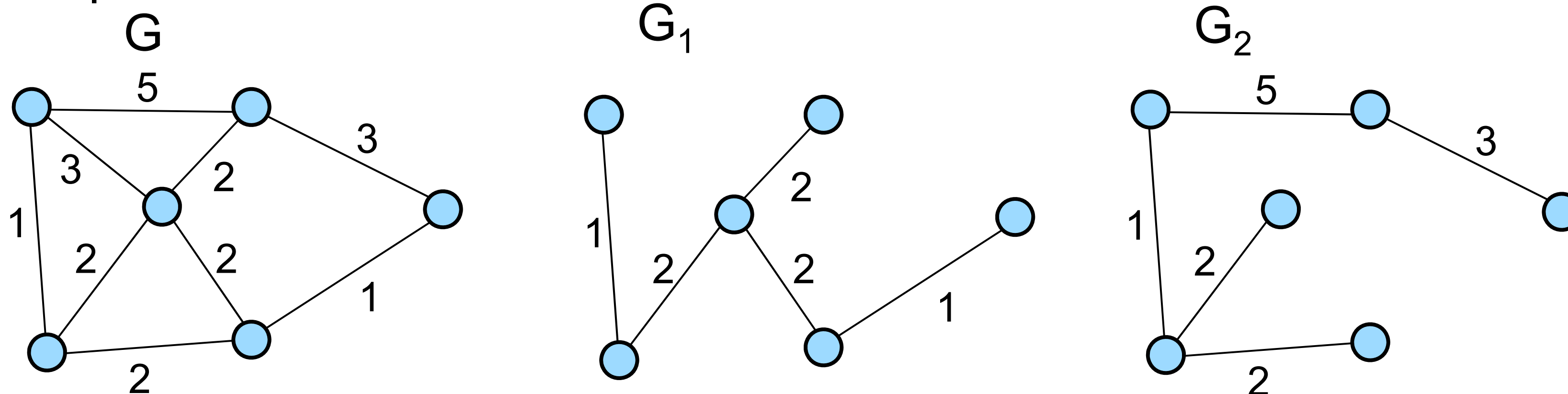
- Wenn G ein Graph ist, dann heißt G‘ aufspannender Baum (oder Spannbaum) von G, wenn
  - G‘ hat gleiche Knotenmenge wie G
  - G‘ enthält Teilmenge der Kantenmenge von G
  - G‘ ist Baum
- Beispiel:



  - G_1 und G_2 sind aufspannende Bäume von G

Definition (II): Minimal aufspannender Baum

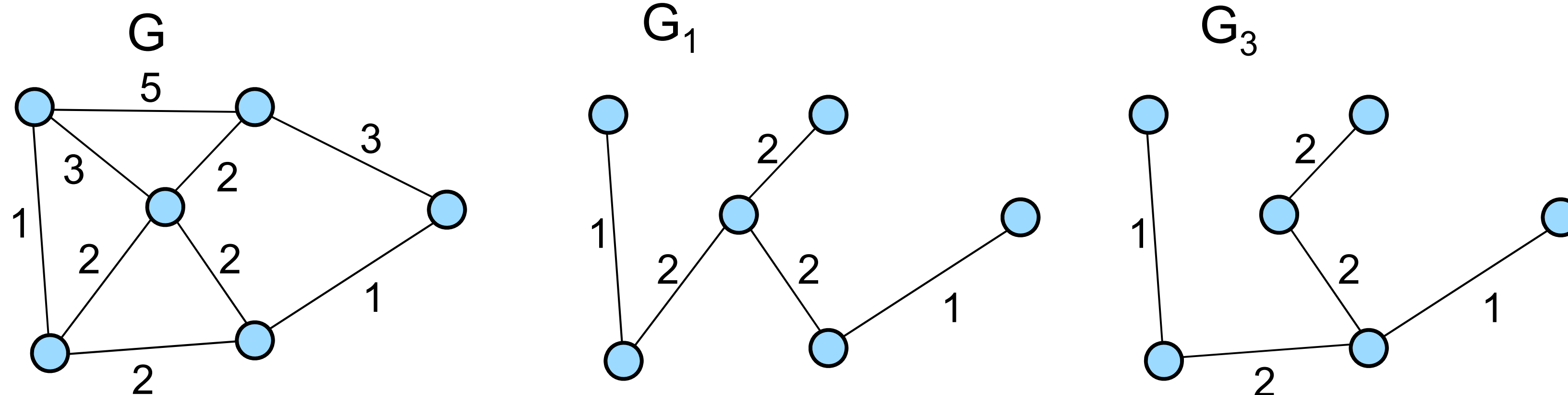
- Wenn G kantenbewerteter Graph, dann heißt G‘ minimal aufspannender Baum (oder minimaler Spannbaum) von G, wenn
  - G‘ aufspannender Baum von G
  - Kantensumme in G‘ kleiner gleich Kantensumme jedes beliebigen aufspannenden Baumes von G
- Beispiel:



  - G_1 und G_2 sind aufspannende Bäume von G
  - G_1 ist minimal aufspannender Baum von G

Nicht-Eindeutigkeit

- Minimal aufspannende Bäume müssen nicht eindeutig sein
- Beispiel:



- G_1 und G_3 sind minimal aufspannende Bäume von G

Algorithmus von Prim

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

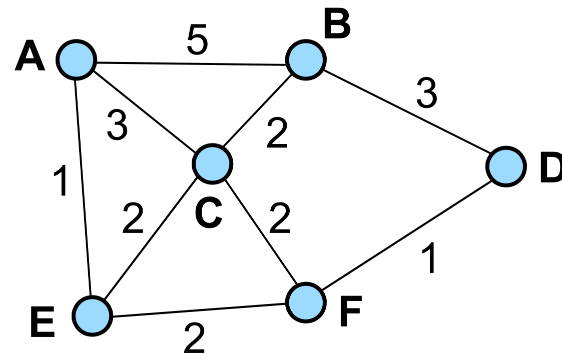
- Prinzip:
  - Kantenbewerteter Graph G = (V,E,g), Startknoten v_s
  - Notation: $\{v_1,v_2\}_x$ g.d.w. $\{v_1,v_2\} \in V$ und $g(\{v_1,v_2\})=x$
  - Minimal aufspannende Baum G'= (V',E',g') wird schrittweise aufgebaut
  - Menge offener Kanten E_{open}: Teilmenge der Kanten in G, die von einem Knoten aus $e \in V'$ zu einem Knoten in $f \in V \setminus V'$ führen
- Algorithmus:

```
( 1) insert v_s into V';
( 2) while (|V'| < |V|){
( 3)   calculate E_open;
( 4)   select {e,f} ∈ E_open with minimal weight;
( 5)   insert f into V';
( 6)   insert {e,f} into E';
( 7) }
```

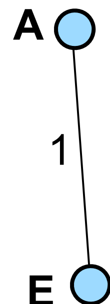
- Anmerkung:
  - Unterschiedliche minimal aufspannende Bäume erhält man in Abhängigkeit von der Wahl der Startknotens und von der Wahl der offenen Kante bei mehreren Kandidaten mit minimalem Gewicht

Beispiel (I)

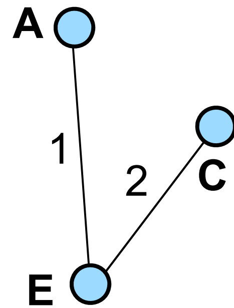
- Ausgangsgraph (Startknoten sei A):

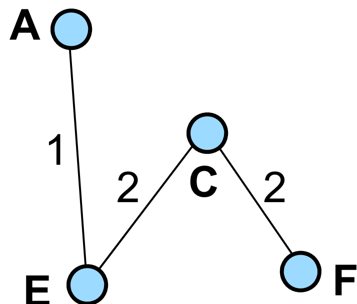


- Durchgang 1:
  - A
  - Menge der offenen Kanten: $\{\{A,B\}_5, \{A,C\}_3, \{A,E\}_1\}$
  - Ausgewählt wird Kante {A,E}
  -

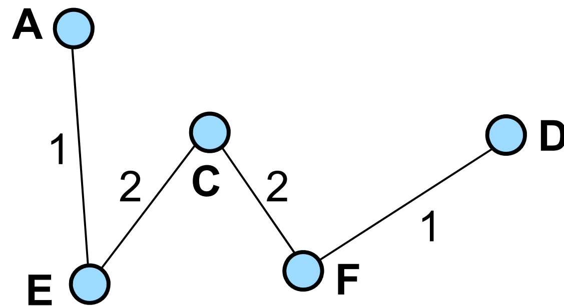


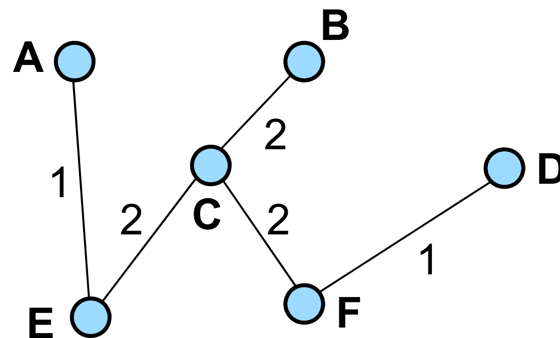
Beispiel (II)

- Durchgang 2:
  - Menge der offenen Kanten: $\{\{A,B\}_5, \{A,C\}_3, \{E,C\}_2, \{E,F\}_2\}$
  - Ausgewählt wird Kante {E,C}
  - 

- Durchgang 3:
  - Menge der offenen Kanten: $\{\{A,B\}_5, \{E,F\}_2, \{C,F\}_2, \{C,B\}_2\}$
  - Ausgewählt wird Kante {C,F}
  - 

Beispiel (III)

- Durchgang 4:
  - Menge der offenen Kanten: $\{\{A,B\}_5, \{C,B\}_2, \{F,D\}_1\}$
  - Ausgewählt wird Kante {F,D}
  - 

- Durchgang 5:
  - Menge der offenen Kanten: $\{\{A,B\}_5, \{C,B\}_2, \{D,B\}_3\}$
  - Ausgewählt wird Kante {C,B}
  - 

  - Algorithmus terminiert, da V = V‘

Komplexität

- n-1 Iterationen (Schleife Zeile 2f.)
- Pro Durchgang wird ein Knoten in Baum eingefügt $\Rightarrow$ O(n)
- In jedem dieser Durchgänge:
  - Bestimmung offener Kanten (Zeile 3)
  - Auswahl minimaler Kante (Zeile 4)
  - Komplexität O(n)
- Schachtelung beider Teile ergibt $O(n^2)$

- Anmerkung:
  - Verbesserung Komplexität durch andere Datenstruktur (Vorrang-Warteschlange)

Algorithmus von Kruskal

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

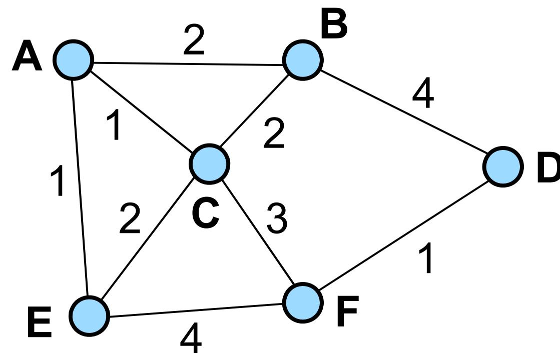
- Prinzip:
  - Kantenbewerteter Graph G = (V,E,g)
  - Notation: $\{v_1,v_2\}_x$ g.d.w. $\{v_1,v_2\} \in V$ und $g(\{v_1,v_2\})=x$
  - Minimaler Spannbaum G'= (V',E',g'):
    - Anfangs nur Knoten (V':= V)
    - E_{sorted}: Kantenliste sortiert nach Markierung
    - Füge sukzessive Kante mit minimalem Gewicht hinzu
- Algorithmus:

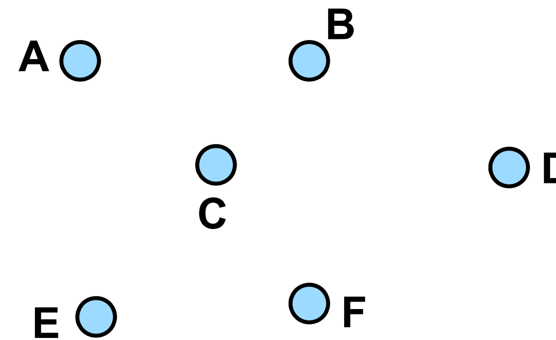
```
( 1) E_sorted := Sort(E,g);
( 2) V' := V;
( 3) while (G' not connected){
( 4)   e_actual := Edge from E_sorted with minimal weight;
( 5)   E_sorted := E_sorted \ e_actual;
( 6)   if ((V',E' ∪ (e_1,e_2),g') has cycle)
( 7)     doNothing;
( 8)   else
( 9)     G' := (V',E' ∪ (e_1,e_2),g');
(10) }
```

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

Beispiel (I)

- Ausgangsgraph und initialer Spannbaum:



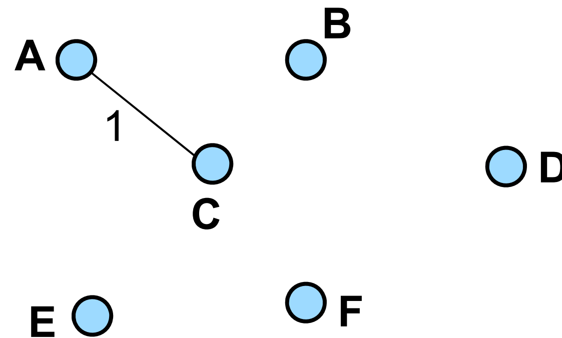


- Kantenliste E_{Sorted}:

Kante	Gewicht
(A,C)	1
(A,E)	1
(D,F)	1
(A,B)	2
(B,C)	2
(C,E)	2
(C,F)	3
(B,D)	4
(E,F)	4

Beispiel (II)

- Wähle Kante mit minimalem Gewicht: $(A,C)_1$
- Kante erzeugt keinen Zyklus, wird hinzugefügt

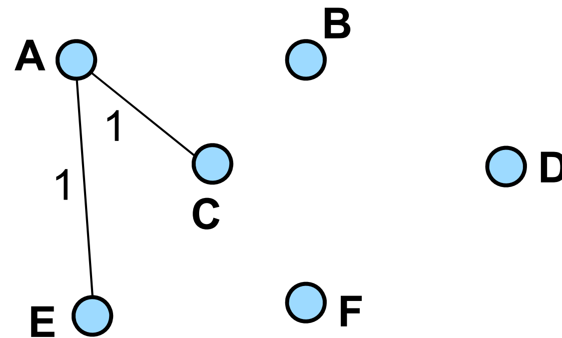


- Aktualisiere Kantenliste E_{sorted}:

Kante	Gewicht
~~(A,C)~~	~~1~~
(A,E)	1
(D,F)	1
(A,B)	2
(B,C)	2
(C,E)	2
(C,F)	3
(B,D)	4
(E,F)	4

Beispiel (III)

- Wähle Kante mit minimalem Gewicht: $(A,E)_1$
- Kante erzeugt keinen Zyklus, wird hinzugefügt

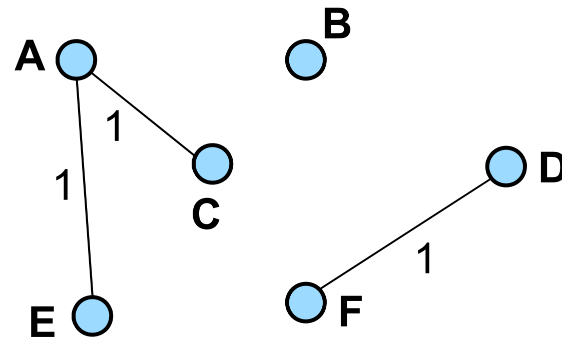


- Aktualisiere Kantenliste E_{sorted}:

Kante	Gewicht
~~(A,E)~~	~~1~~
(D,F)	1
(A,B)	2
(B,C)	2
(C,E)	2
(C,F)	3
(B,D)	4
(E,F)	4

Beispiel (IV)

- Wähle Kante mit minimalem Gewicht: $(D,F)_1$
- Kante erzeugt keinen Zyklus, wird hinzugefügt

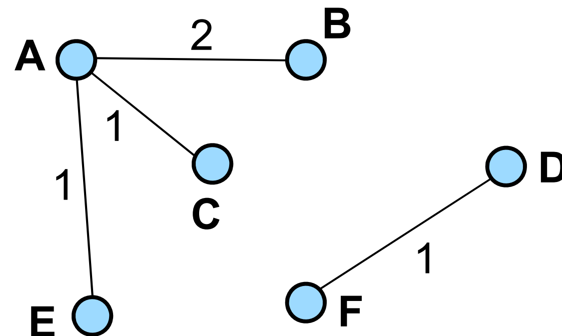


- Aktualisiere Kantenliste E_{sorted}:

Kante	Gewicht
~~(D,F)~~	~~1~~
(A,B)	2
(B,C)	2
(C,E)	2
(C,F)	3
(B,D)	4
(E,F)	4

Beispiel (V)

- Wähle Kante mit minimalem Gewicht: $(A,B)_2$
- Kante erzeugt keinen Zyklus, wird hinzugefügt



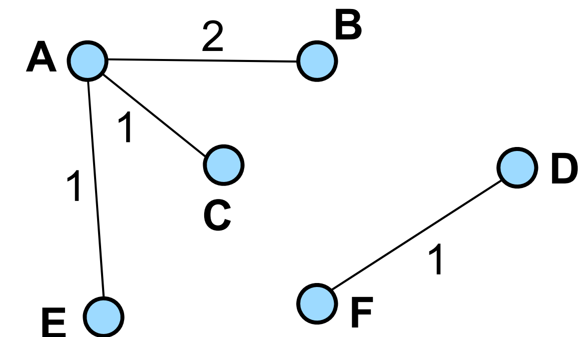
- Aktualisiere Kantenliste E_{sorted}:

Kante	Gewicht
~~(A,B)~~	~~2~~
(B,C)	2
(C,E)	2
(C,F)	3
(B,D)	4
(E,F)	4

Beispiel (VI)

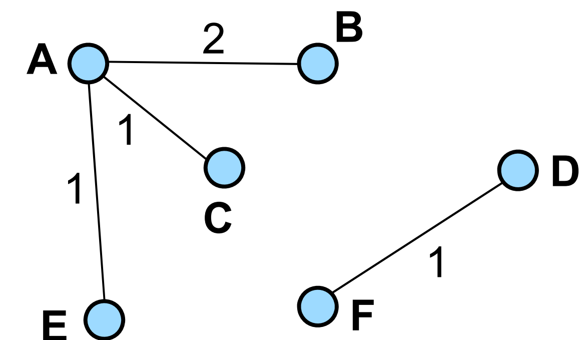
- Wähle Kante mit minimalem Gewicht: $(B,C)_2$
- Kante erzeugt Zyklus, wird verworfen
- Aktualisiere Kantenliste E_{sorted}:

Kante	Gewicht
~~(B,C)~~	~~2~~
(C,E)	2
(C,F)	3
(B,D)	4
(E,F)	4



- Wähle Kante mit minimalem Gewicht: $(C,E)_2$
- Kante erzeugt Zyklus, wird verworfen
- Aktualisiere Kantenliste E_{sorted}:

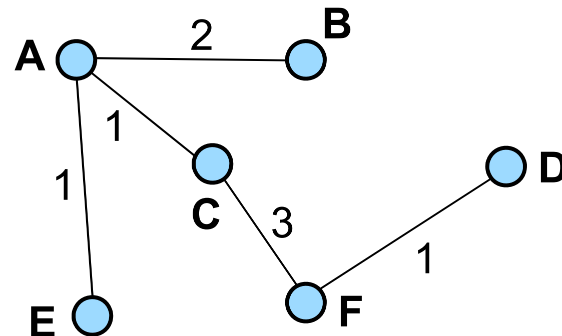
Kante	Gewicht
~~(C,E)~~	~~2~~
(C,F)	3
(B,D)	4
(E,F)	4



Beispiel (VII)

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

- Wähle Kante mit minimalem Gewicht: $(C,F)_3$
- Kante erzeugt keinen Zyklus, wird hinzugefügt



- Alle Knoten verbunden ⇒ Algorithmus terminiert

Komplexität / Anmerkung

- Wichtige Operationen:
  - Find: Herausfinden, ob Zyklus entsteht
  - Union: Vereinigen zweier Bäume
- Pro Kante:
  - Maximal zwei find-Operationen
  - Maximal eine union-Operationen
- Insgesamt: $O(n + e \cdot \log_2 n)$ mit e Anzahl Kanten
- Hinweis: Setzt „gute“ Datenstruktur und „geschicktes“ Zusammenfügen der Bäume voraus

- Anmerkung:
  - Unterschiedliche minimal aufspannende Bäume entstehen in Abhängigkeit der Kantenwahl in Zeile (4) bei mehreren Kanten mit minimalem Gewicht

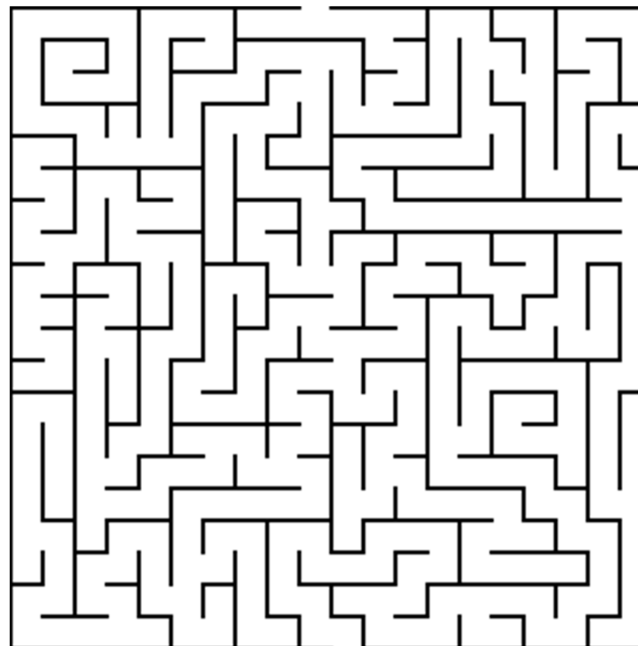
Anwendungen (I)

- Kommunikationsnetze (siehe Motivation):
  - Telefonnetze
  - Elektrische Netze
- Computernetzwerke:
  - STP (Spanning Tree Protocol):
    - Standard in geswitchten Umgebungen (IEEE-Norm 802.1D, 1998)
    - Zentraler Teil von Switch-Infrastrukturen
    - Rechnernetz mit Vielzahl von Switches
    - STP stellt sicher, dass zwischen zwei Rechnern eindeutiger Datenpfad
  - Modifizierte Version RSTP (Rapid STP), (IEEE-Norm 802.1D, 2004):
    - Bei signalisierten Topologieänderungen:
      - Keine Löschung, sondern Weiterarbeiten wie bisher
      - Alternativpfade berechnen
    - Damit: Deutliche Reduzierung Ausfallzeit

Anwendungen (II)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Spieleprogrammierung:
  - Labyrintherzeugung
  - Knoten entspricht Feld, Kante Übergang zum Nachbarfeld
  - Also: Fehlende Kante entspricht Wand
  - Mittels Spannbaum erzeugtes Labyrinth besitzt nur einen Lösungsweg



[http://www.mazegenerator.net/]

Übersicht

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

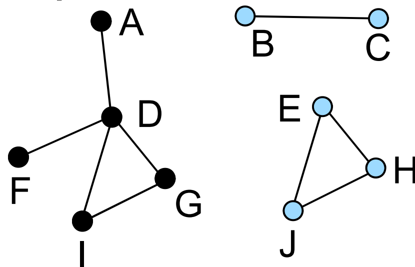
- Transitiver Abschluss
- Minimal aufspannende Bäume
- Suchverfahren in Graphen
- Breitensuche
- Tiefensuche

Suchverfahren in Graphen (I)

- Klasse von Verfahren zum systematischen Durchlaufen aller Knoten/Kanten eines Graphen
- Problem tritt z.B. auf bei Fragestellung
  - ob ein ungerichteter Graph zusammenhängend ist
  - ob ein gerichteter Graph einen Zyklus enthält
- Grundlegende Idee für diese Algorithmen:
  - Beginne an einem bestimmten Startknoten v_s
  - Besuche alle von v_s erreichbaren Knoten
  - Verlauf der Suche spiegelt sich im sog. Erreichbarkeitsbaum EB wider
  - EB ist ein Wurzelbaum mit v_s als Wurzel
  - EB besteht aus den von v_s aus erreichbaren Knoten und den Kanten, die zu diesen Knoten führen
  - Zu einem v_s können verschiedene EB existieren (Gleiche Knotenmenge, aber verschiedene Kantenmenge)

Suchverfahren in Graphen (II)

- Basis aller Suchverfahren ist folgender Markierungsalgorithmus (Name kommt daher, dass bereits besuchte Knoten markiert werden):
  (1) Markiere den Startknoten
  (2) Solange noch Kanten von markierten zu unmarkierten Knoten existieren, wähle eine solche Kante aus und markiere den Endknoten dieser Kante
- Beispiel:



Welche Knoten sind von A aus erreichbar?

- Markiere A (Schritt (1))
- Sind noch Kanten von markierten zu unmarkierten Knoten vorhanden? JA: {A,D}
- Wähle {A,D} und markiere D
- Sind noch Kanten von markierten zu unmarkierten Knoten vorhanden? JA: {D,F}, {D,I}, {D,G}
- Wähle {D,G} und markiere G
- Sind noch Kanten von markierten zu unmarkierten Knoten vorhanden? JA: {D,F}, {D,I}, {G,I}
- Wähle {D,I} und markiere I
- Sind noch Kanten von markierten zu unmarkierten Knoten vorhanden? JA: {D,F}
- Wähle {D,F} und markiere F
- Sind noch Kanten von mark. zu unmark. Knoten vorhanden? NEIN $\Rightarrow$ Ende

Suchverfahren in Graphen (III)

- Eigenschaften des Verfahrens:
  - Die verwendeten Kanten bilden am Ende den EB
  - Da in jedem Schritt ein Knoten markiert wird, folgt aus der Endlichkeit des Graphen die Terminierung des Algorithmus
  - Menge der am Ende markierten Knoten ist die Menge der erreichbaren Knoten
  - Bei entsprechender Realisierung der Auswahl in Schritt (2) hat der Algorithmus eine Laufzeit von O(n+m)
  - Algorithmus kann durch Modifizierung die Frage der Erreichbarkeit eines Knoten von s aus in n Schritten beantworten
  - Alle Suchverfahren basieren auf dieser Grundidee, unterscheiden sich jedoch in der Art und Weise, wie in Schritt (2) Kanten ausgewählt werden

Suchverfahren in Graphen (IV)

- Tiefensuche (depth-first-search):
  - Versucht, den am weitesten vom Startknoten entfernten Knoten so früh wie möglich zu besuchen
  - Dazu wählt man in Schritt (2) eine Kante aus, deren Startknoten der zuletzt markierte Knoten ist
  - Sind alle von diesem Knoten ausgehenden Kanten schon abgearbeitet, dann arbeite die markierten Knoten in umgekehrter Reihenfolge ab
  - Herkunft Verfahrensnamen: Man geht bei jedem Schritt „so tief wie möglich in den Graphen hinein“
- Breitensuche (breadth-first-search):
  - Suche wird so breit wie möglich angelegt, d.h. für jeden besuchten Knoten werden zunächst alle Nachbarn besucht
  - Dazu wählt man in Schritt (2) jeweils eine Kante, deren Startknoten der am längsten markierte Knoten ist

Übersicht

- Transitiver Abschluss
- Minimal aufspannende Bäume
- Suchverfahren in Graphen
- Breitensuche
- Tiefensuche

Breitensuche (I)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Idee: Bearbeite Knoten, der in n Schritten von u erreichbar ist, erst dann, wenn alle in n-1 Schritten erreichbaren Knoten bereits abgearbeitet worden sind
- Eingabe: Ungerichteter Graph G = (V,E), Startknoten $v_s \in V$
- Als Hilfsstruktur wird eine Schlange von Knoten benötigt:
  - Schlange ist Struktur von prinzipiell unendlicher Kapazität
  - Arbeitet nach dem FIFO-Prinzip (First In First Out), d.h. es kann jeweils nur das älteste Element entnommen werden
  - Funktionen auf dem Datentyp Schlange:
    - `put(v)`: Lege den Knoten v in der Schlange ab
    - `get()`: Entnehme Knoten aus der Schlange
    - `read()`: Lesen aus der Schlange ohne zu Entnehmen
    - `isEmpty()`: Prüft, ob die Schlange leer ist oder nicht
- Als weitere Hilfsstruktur werden zu jedem Knoten aktueller Farbwert, Abstand d zu Startknoten v_s und der Vorgänger pred verwaltet, von dem aus der Knoten erreicht worden ist

Breitensuche (II)

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

- Farbwerte:
  - weiß = unbehandelt
  - grau = in Schlange
  - schwarz = abgearbeitet
- Funktion `succ(v)` liefert die Menge der direkten Nachbarn (bzw. direkten Nachfolger im gerichteten Fall) des Knotens v
- Resultat:
  - Aus Distanzwerten kann am Ende das Ergebnis der Breitensuche abgelesen werden
  - Ist G zusammenhängend, liefern pred-Werte einen Spannbaum
  - Ist G nicht zusammenhängend, liefern pred-Werte einen Spannwald

Algorithmus (I)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Sei v_s Startknoten, Q Schlange

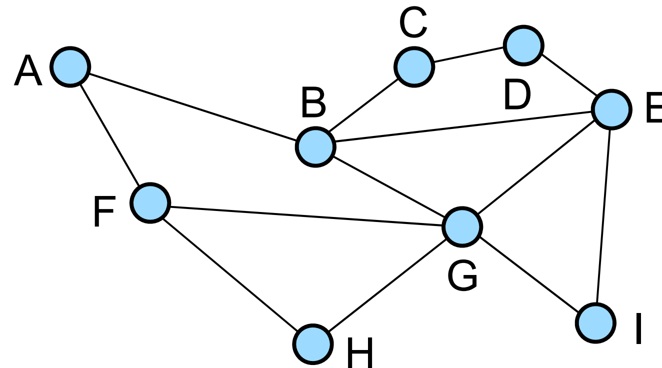
```
( 1) // Initialisierungen
( 2) for each v ∈ V/{vs} {
( 3)   farbe[v] = weiß;
( 4)   d[v] = ∝;
( 5)   pred[v] = null;
( 6) }
( 7) farbe[vs] = grau;
( 8) d[vs] = 0;
( 9) Q.put(vs);
...
```

Algorithmus (II)

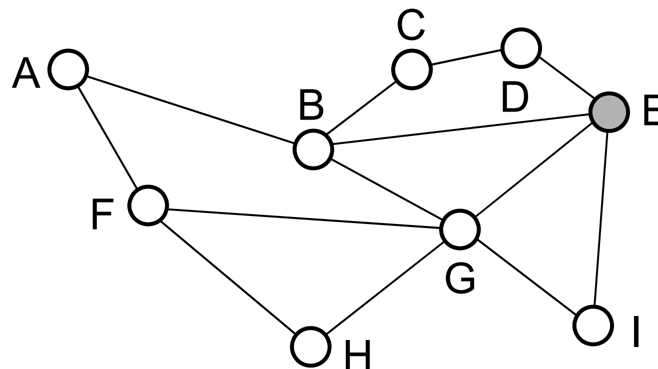
```
...
(10) // Schleife zum Abarbeiten der Schlange
(11) while (!Q.isEmpty()){
(12)   v = Q.read();
(13)   for each u ∈ succ(v){
(14)      if (farbe[u] == weiß){
(15)        // Unbearbeitete Knoten in Schlange einfügen
(16)        farbe[u] = grau;
(17)        d[u] = d[v] + 1;
(18)        pred[u] = v;
(19)        Q.put(u);
(20)      }
(21)   }
(22)   Q.get();
(23)   farbe[v] = schwarz;
(24) }
```

Beispiel (I)

- Beispiel: Breitensuche in folgendem Graphen mit Startknoten E



- Initialisierung:

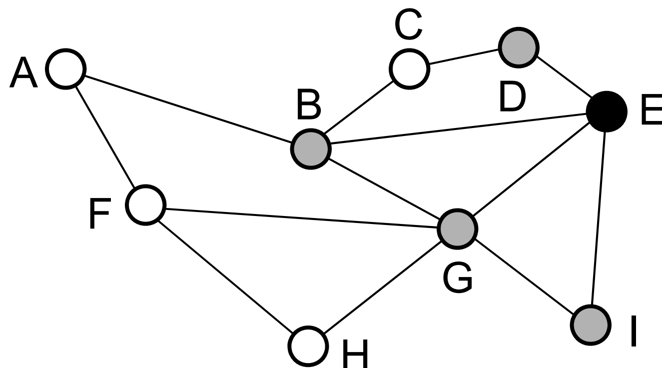


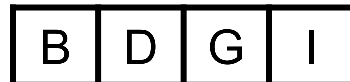
E

Knoten	d	pred
A	∞	null
B	∞	null
C	∞	null
D	∞	null
E	0	null
F	∞	null
G	∞	null
H	∞	null
I	∞	null

Beispiel (II)

- Durchgang 1:
  - v = E
  - Alle unbearbeiteten (d.h. weißen) Nachbarn von E: Grau färben, Distanz eintragen, E als Vorgänger eintragen und in Schlange eintragen
  - E aus Schlange entnehmen und schwarz färben



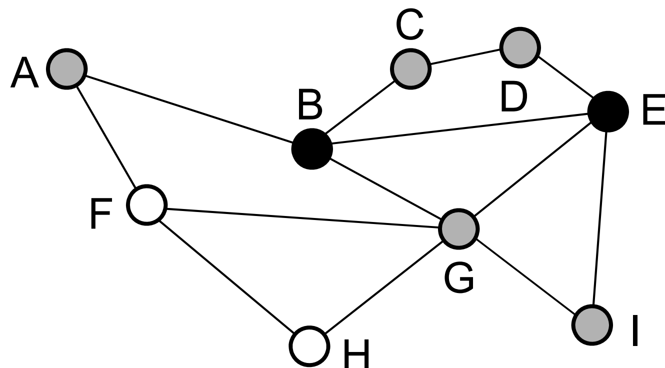


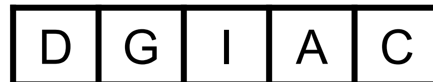
Knoten	d	pred
A	∞	null
B	1	E
C	∞	null
D	1	E
E	0	null
F	∞	null
G	1	E
H	∞	null
I	1	E

  - Schlange nicht leer ⇒ Weitermachen

Beispiel (III)

- Durchgang 2:
  - v = B
  - Alle unbearbeiteten (d.h. weißen) Nachbarn von B: Grau färben, Distanz eintragen, B als Vorgänger eintragen und in Schlange eintragen
  - B aus Schlange entnehmen und schwarz färben



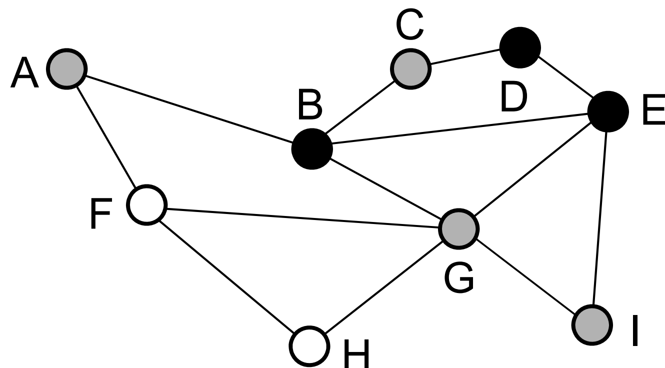


Knoten	d	pred
A	2	B
B	1	E
C	2	B
D	1	E
E	0	null
F	∞	null
G	1	E
H	∞	null
I	1	E

  - Schlange nicht leer $\Rightarrow$ Weitermachen

Beispiel (IV)

- Durchgang 3:
  - v = D
  - Alle unbearbeiteten (d.h. weißen) Nachbarn von D: Grau färben, Distanz eintragen, B als Vorgänger eintragen und in Schlange eintragen
  - Entfällt, da diese Menge leer ist
  - D aus Schlange entnehmen und schwarz färben

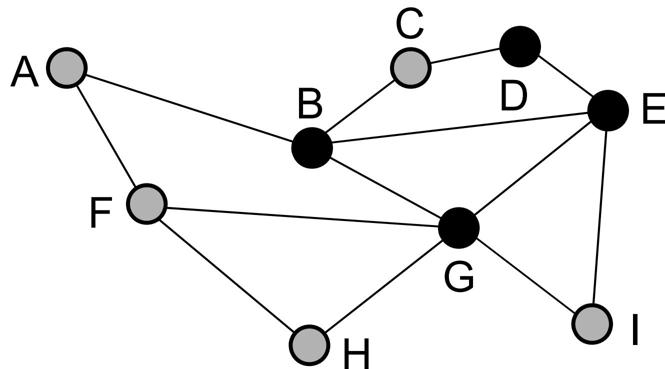


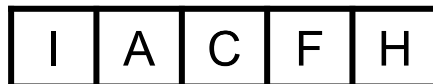
  - Schlange nicht leer $\Rightarrow$ Weitermachen

Knoten	d	pred
A	2	B
B	1	E
C	2	B
D	1	E
E	0	null
F	∞	null
G	1	E
H	∞	null
I	1	E

Beispiel (V)

- Durchgang 4:
  - v = G
  - Alle unbearbeiteten (d.h. weißen) Nachbarn von G: Grau färben, Distanz eintragen, G als Vorgänger eintragen und in Schlange eintragen
  - G aus Schlange entnehmen und schwarz färben



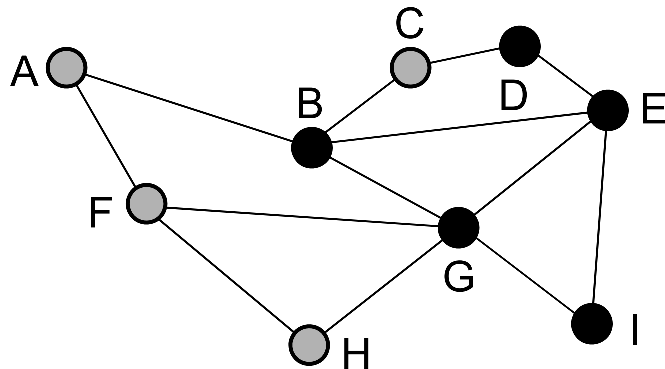


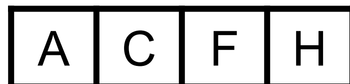
Knoten	d	pred
A	2	B
B	1	E
C	2	B
D	1	E
E	0	null
F	2	G
G	1	E
H	2	G
I	1	E

  - Schlange nicht leer $\Rightarrow$ Weitermachen

Beispiel (VI)

- Durchgang 5:
  - v = I
  - Alle unbearbeiteten (d.h. weißen) Nachbarn von I: Grau färben, Distanz eintragen, G als Vorgänger eintragen und in Schlange eintragen
  - Entfällt, da die Menge leer ist
  - I aus Schlange entnehmen und schwarz färben



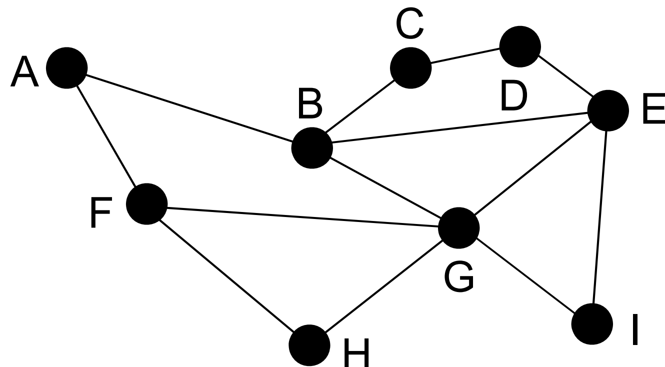


  - Schlange nicht leer ⇒ Weitermachen

Knoten	d	pred
A	2	B
B	1	E
C	2	B
D	1	E
E	0	null
F	2	G
G	1	E
H	2	G
I	1	E

Beispiel (VII)

- Durchgang 6-9:
  - Analog zu Durchgang 5 mit v = A bzw. C bzw. F bzw. H
  - Schritte im Schleifenrumpf entfallen jeweils, da keine unmarkierten Knoten mehr
  - A bzw. C bzw. F bzw. H werden aus Schlange entnommen und schwarz gefärbt



  - Schlange leer $\Rightarrow$ Algorithmus terminiert

Knoten	d	pred
A	2	B
B	1	E
C	2	B
D	1	E
E	0	null
F	2	G
G	1	E
H	2	G
I	1	E

Beispiel (VIII) / Komplexität

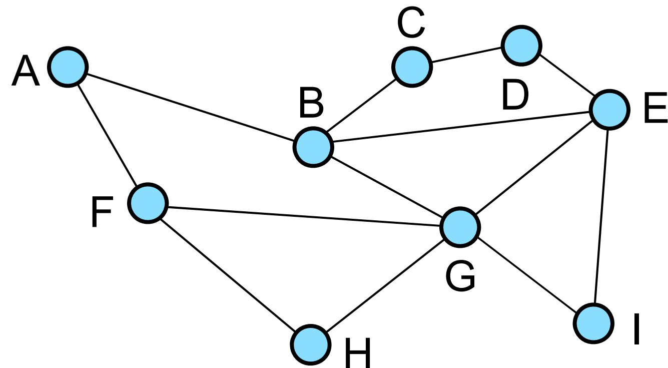
DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Resultat:
  - Aus der Spalte d kann das Resultat direkt abgelesen werden
  - Es gibt i.A. mehrere Lösungen, da mehrere Knoten mit einer bestimmten Distanz vom Startknoten aus erreichbar sind
  - Bsp.: E B D G I A C F H

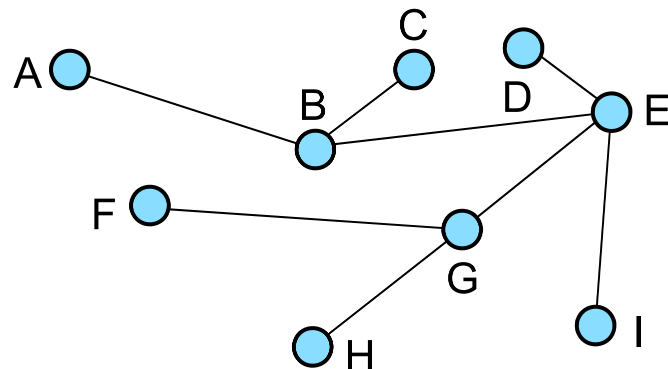
- Komplexität der Breitensuche:
  - Ein Besuch pro Knoten und Kante, d.h. O(n+m)
  - Falls G zusammenhängend, gilt $|E| > |V|-1 \Rightarrow$ Komplexität O(m)

Breitensuche/Spannbaum

- Nimmt man die ausgewählten Kanten (d.h. die am Ende in der Tabelle stehen), so ist das Resultat der Breitensuche ein Spannbaum
- Im Beispiel ergibt sich zum Graphen



der Spannbaum

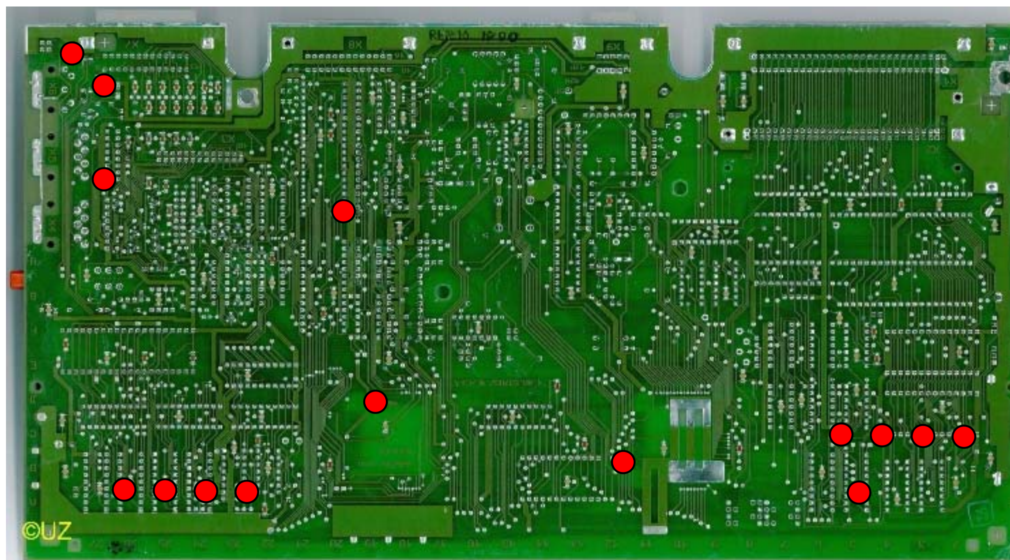


Knoten	d	pred
A	2	B
B	1	E
C	2	B
D	1	E
E	0	null
F	2	G
G	1	E
H	2	G
I	1	E

Anwendungen (I)

- Distanzprobleme (Kürzester unbewerteter Pfad)
  - z.B. Bearbeitung von Platinen oder Holzplatten
  - Löt- oder Bohrkopf soll über Platte fahren und dabei unterschiedliche Löt- bzw. Bohrstellen anfahren
  - Kosten für Bewegung zwischen zwei Stellen gleich
  - Durch Breitensuche wird dabei minimaler Weg ermittelt

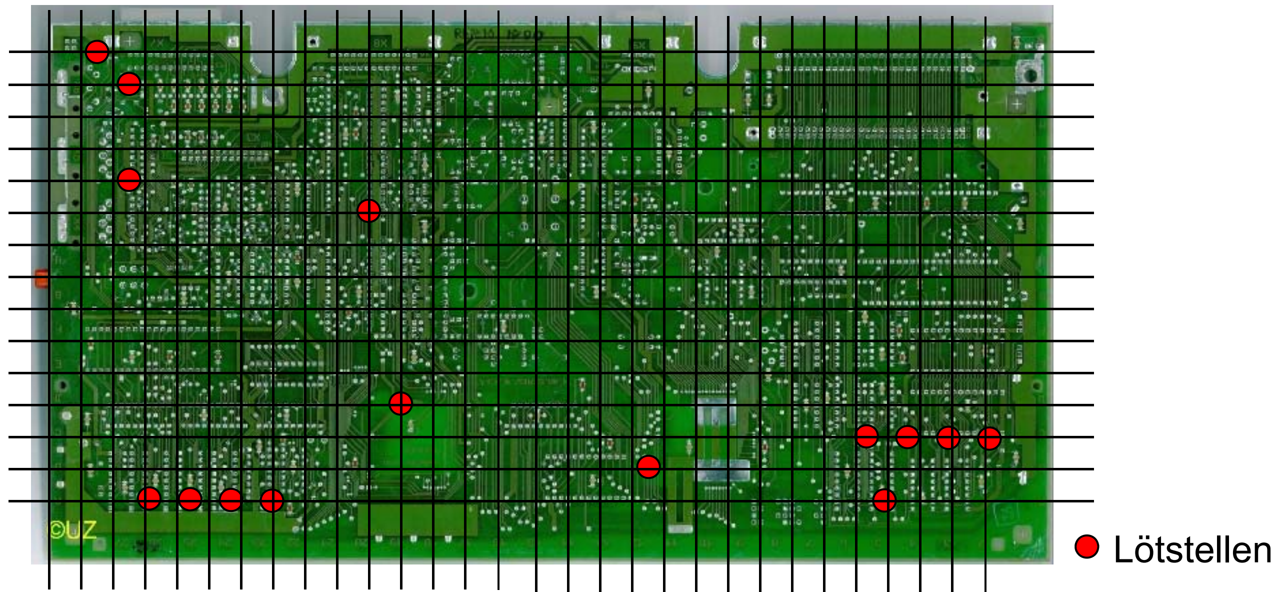
[http://www.mec-elektronik.de/]



● Lötstellen

[http://greigo.com/computer-platine#]

Anwendungen (II)

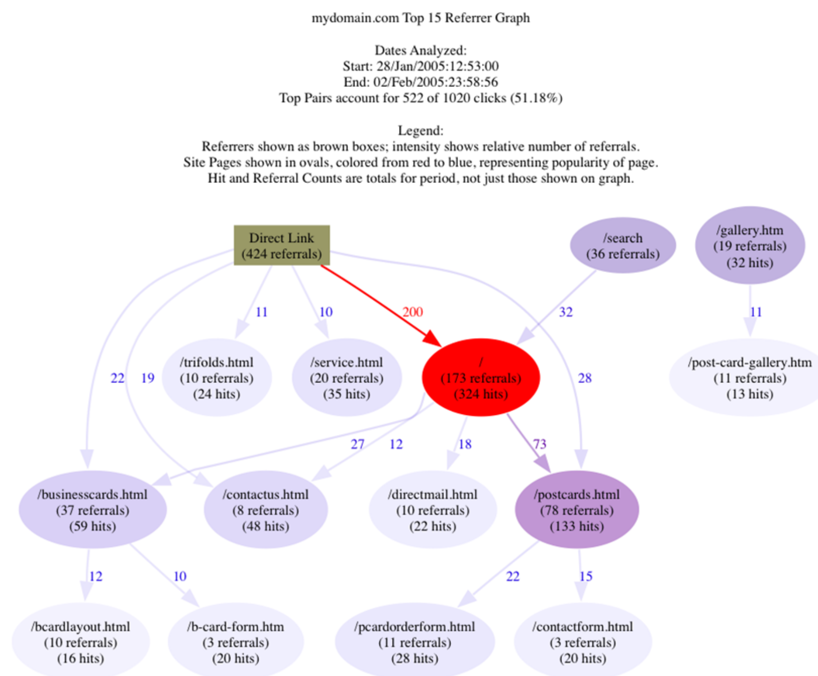


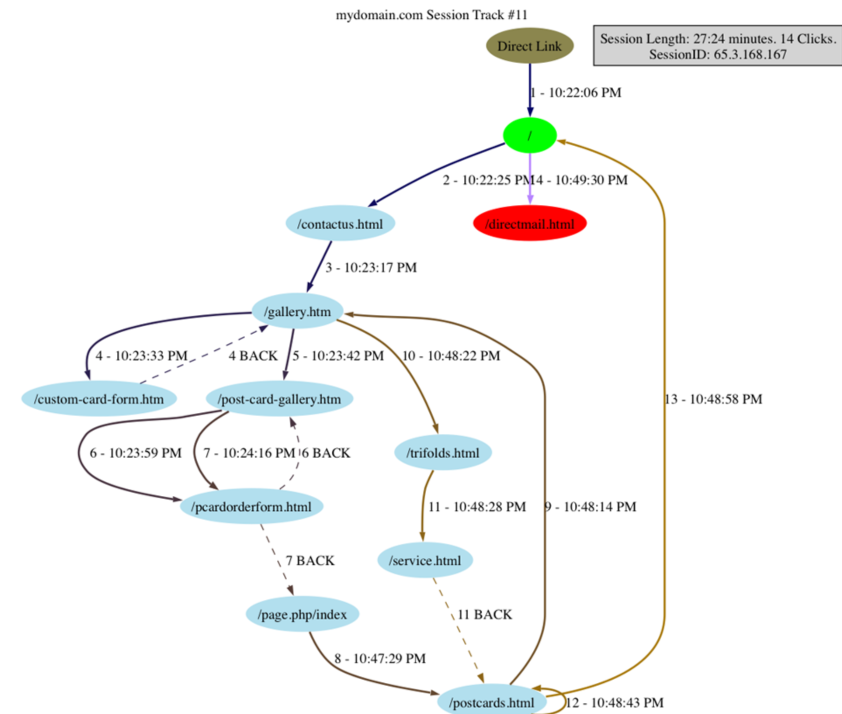
- Erläuterung:
  - Lege Gitter über Platine und markiere Lötstellen:
  - Bohrkopf verarbeitet Startknoten
  - Dann alle Knoten mit Abstand 1 zum Startknoten
  - Dann alle Knoten mit Abstand 2 zum Startknoten
  - usw.

- Auffinden aller Knoten innerhalb einer Zusammenhangskomponente

Anwendungen (III)

- Click-Stream-Analyse
  - Weblog: Aufrufreihenfolgen von Seiten durch Benutzer/innen
  - Fragestellungen Weblog-Analyse:
    - In welcher Reihenfolge werden Seiten nacheinander aufgerufen?
    - Wie viele Schritte braucht Nutzer/in durchschnittlich von Seite A zu B?





[http://statviz.sourceforge.net/]

Übersicht

- Transitiver Abschluss
- Minimal aufspannende Bäume
- Suchverfahren in Graphen
- Breitensuche
- Tiefensuche

Algorithmus mit Stapel (I)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

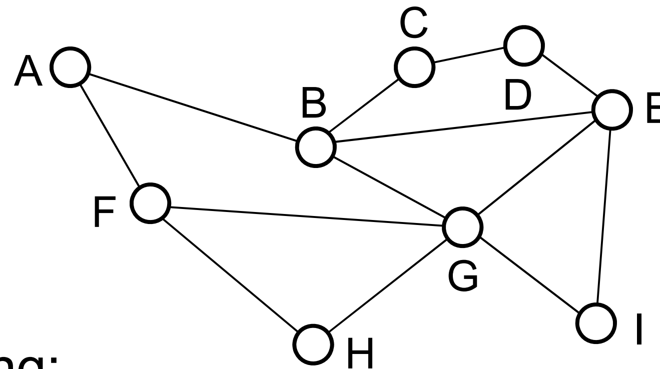
- Gegeben:
  - Ungerichteter Graph G = (V,E)
  - Stapel S zur Verwaltung von Knoten:
    - `push(v)`: Lege Knoten v auf S ab
    - `pop()`: Lese Knoten von S
    - `peek()`: Lese Knoten von S ohne diesen zu entfernen
    - `empty()`: Liefert `true` bei leerem Stapel, `false` sonst
  - Startknoten $v_S \in V$
  - Feld `marked` gibt an, ob Knoten markiert (true) oder nicht (false)
  - Funktion `unmarkedNeighbours(v)` liefert unmarkierte Nachbar von Knoten v
  - Funktion `output(v)` gibt Knoten v aus bzw. verarbeitet diesen

Algorithmus mit Stapel (II)

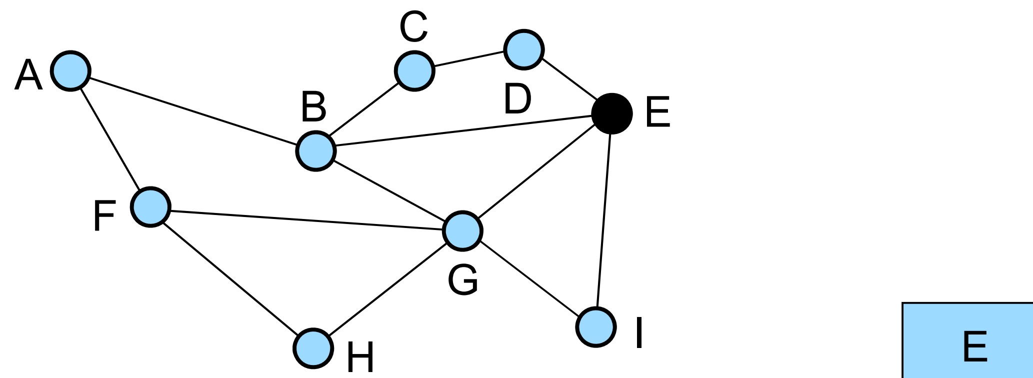
```
( 1) // Initialisierungen
( 2) for each v ∈ V {
( 3)   marked[v] = false;
( 4) }
( 5) marked[vS] = true;
( 6) S.push(vS);
( 7) // Hauptschleife
( 8) while (!S.empty()){
( 9)   actNode = S.pop();
(10)   output(actNode);
(11)   for each v ∈ unmarkedNeighbours(actNode){
(12)     marked[v] = true;
(13)     S.push(v);
(14)   }
(15) }
```

Beispiel (I)

- Tiefensuche in folgendem Graphen mit Startknoten E



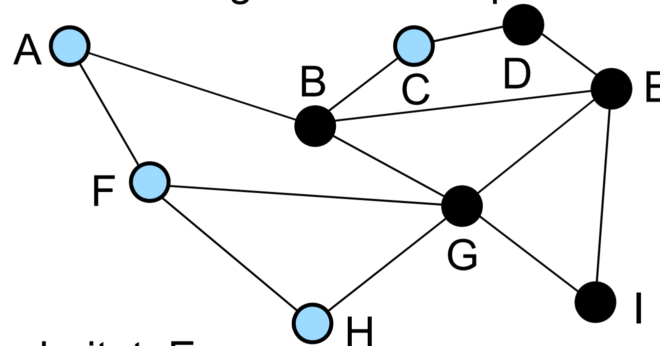
- Initialisierung:
  - Markiere alle Knoten mit `false`
  - Markiere Startknoten E und lege diesen auf den Stapel



  - Bisher besucht: -

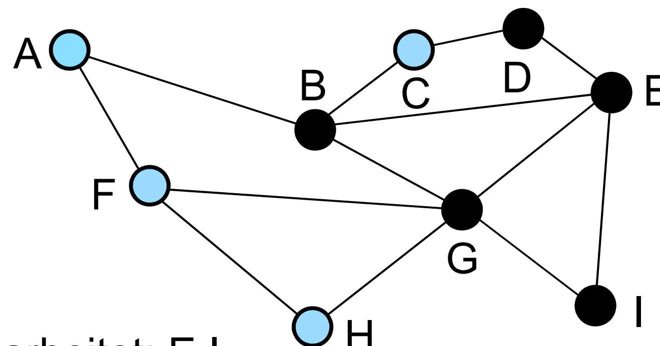
Beispiel (II)

- Durchgang 1:
  - $v_{Aktuell}$ = E, entferne E vom Stapel, bearbeite E
  - Sind noch nicht markierte Nachbarn von E vorhanden? Ja: B, D, G, I
  - Markiere diese und lege sie auf Stapel



I
G
D
B

  - Bisher bearbeitet: E
- Durchgang 2:
  - $v_{Aktuell}$ = I , entferne I vom Stapel, bearbeite I
  - Sind noch nicht markierte Nachbarn von I vorhanden? Nein

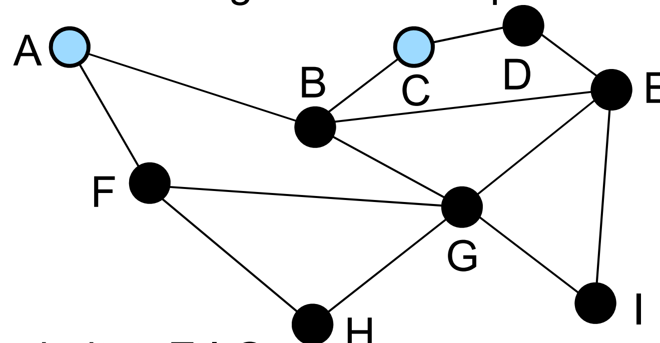


G
D
B

  - Bisher bearbeitet: E I

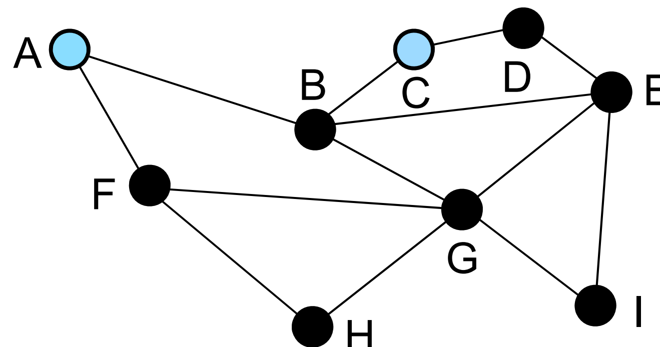
Beispiel (III)

- Durchgang 3:
  - $v_{Aktuell}$ = G, entferne G vom Stapel, bearbeite G
  - Sind noch nicht markierte Nachbarn von G vorhanden? Ja: F, H
  - Markiere diese und lege sie auf Stapel



H
F
D
B

  - Bisher bearbeitet: E I G
- Durchgang 4:
  - $v_{Aktuell}$ = H , entferne H vom Stapel, bearbeite H
  - Sind noch nicht markierte Nachbarn von H vorhanden? Nein

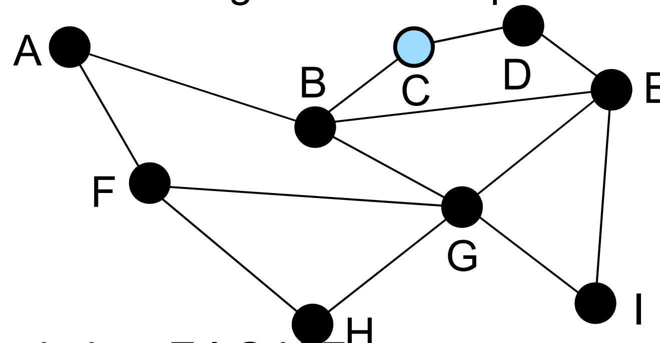


F
D
B

  - Bisher bearbeitet: E I G H

Beispiel (IV)

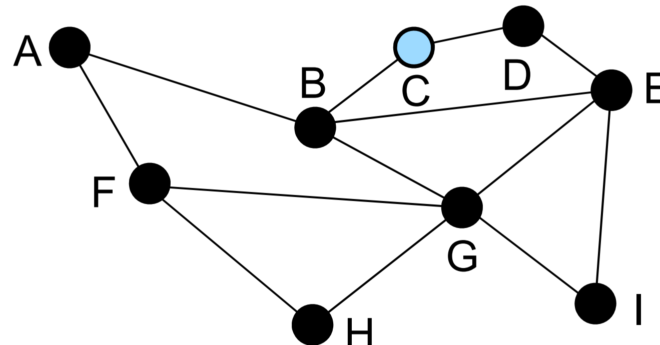
- Durchgang 5:
  - $v_{Aktuell}$ = F, entferne F vom Stapel, bearbeite F
  - Sind noch nicht markierte Nachbarn von F vorhanden? Ja: A
  - Markiere diese und lege sie auf Stapel



A
D
B

  - Bisher bearbeitet: E I G H F

- Durchgang 6:
  - $v_{Aktuell}$ = A , entferne A vom Stapel, bearbeite A
  - Sind noch nicht markierte Nachbarn von A vorhanden? Nein

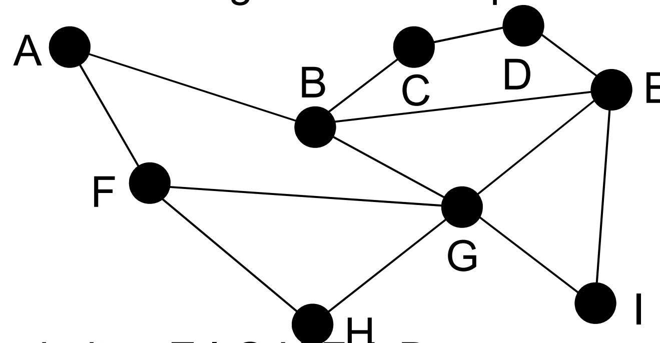


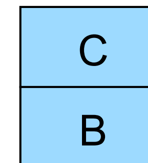
D
B

  - Bisher bearbeitet: E I G H F A

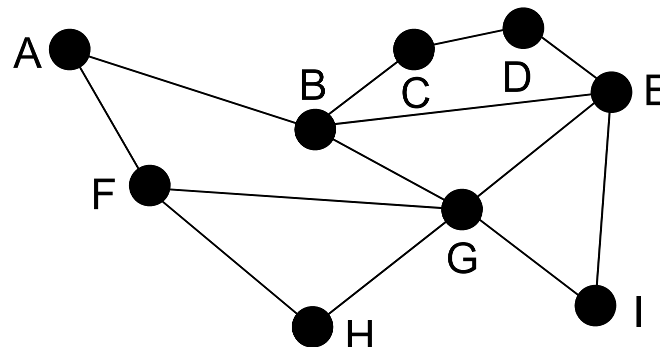
Beispiel (V)

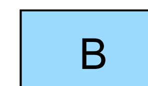
- Durchgang 7:
  - $v_{Aktuell}$ = D, entferne D vom Stapel, bearbeite D
  - Sind noch nicht markierte Nachbarn von D vorhanden? Ja: C
  - Markiere diese und lege sie auf Stapel





  - Bisher bearbeitet: E I G H F A D
- Durchgang 8:
  - $v_{Aktuell}$ = C , entferne C vom Stapel, bearbeite C
  - Sind noch nicht markierte Nachbarn von C vorhanden? Nein

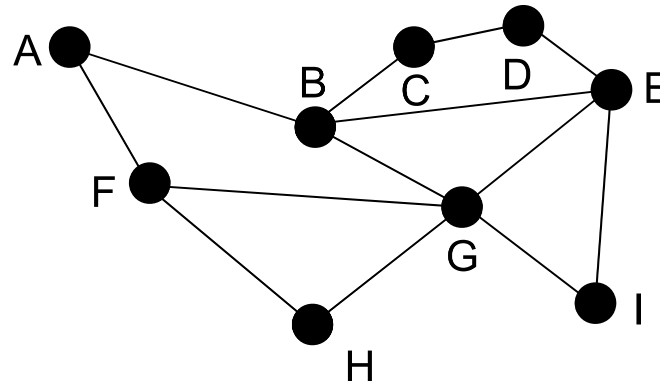




  - Bisher bearbeitet: E I G H F A D C

Beispiel (VI)

- Durchgang 9:
  - $v_{Aktuell}$ = B, entferne B vom Stapel, bearbeite B
  - Sind noch nicht markierte Nachbarn von B vorhanden? Nein



  - Bisher bearbeitet: E I G H F A D C B
- Stapel leer ⇒ Algorithmus terminiert

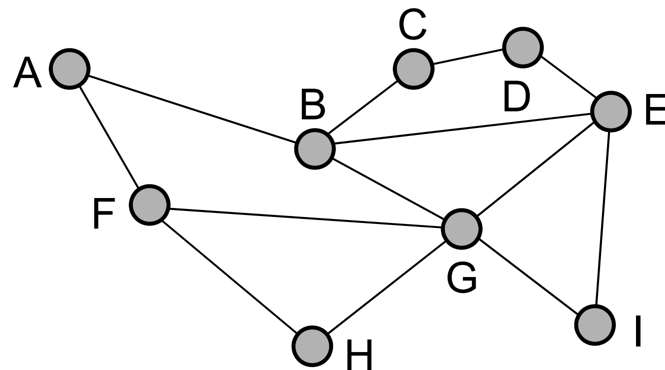
Algorithmus rekursiv

- Gegeben:
  - Ungerichteter Graph G = (V,E)
  - Startknoten $v_s \in V$
  - Feld `marked` gibt an, ob Knoten markiert (true) oder nicht (false)
  - Funktion `neighbours(v)` liefert Menge der Nachbarn von Knoten v

```
( 1) // Initialisierungen
( 2) for each v ∈ V {
( 3)   marked[v] = false;
( 4) }
( 5) depthFirstSearchRecursive(vs);
( 6) // Rekursive Prozedur
( 7) proc depthFirstSearchRecursive(Knoten k){
( 8)   if (!marked[k]){
( 9)     marked[k] = true;
(10)     output(k);
(11)     for each v ∈ neighbours(k){
(12)       depthFirstSearchRecursive(v);
(13)     }
(14)   }
(15) }
```

Beispiel (I)

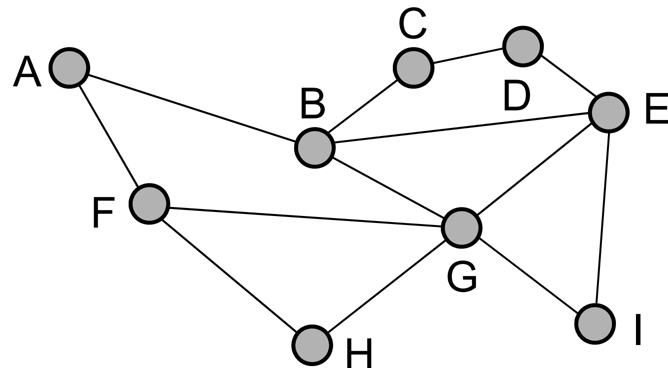
- Tiefensuche in folgendem Graphen mit Startknoten E



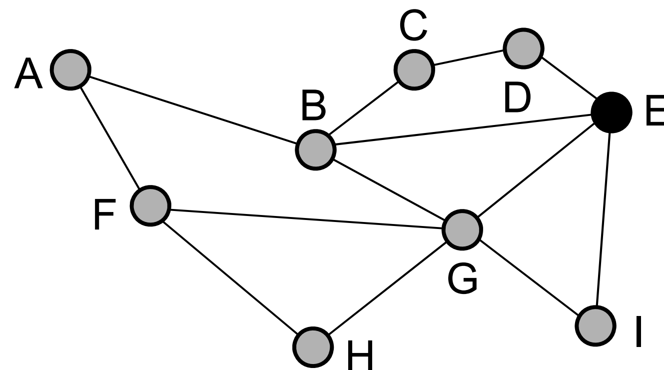
- Annahme:
  - Nachbar eines Knoten in aufsteigender Folge behandeln (`for each`-Schleife in Zeile 11)
  - Markierter Knoten = schwarz
  - Nicht markierter Knoten = grau

Beispiel (II)

- Initialisierung: Markiere alle Knoten mit `false`

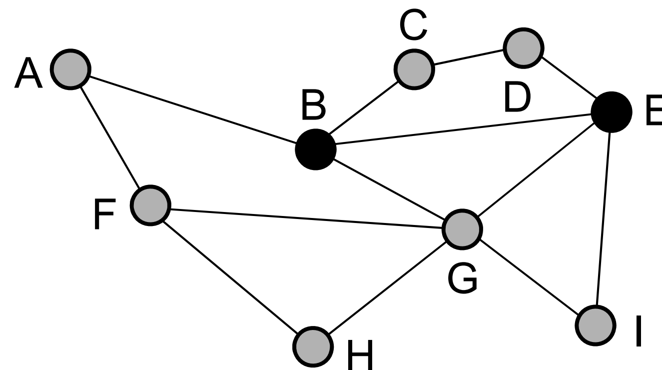


- Durchgang 1: Rufe Prozedur für Startknoten E auf
  - Markiere E
  - Gebe E aus, bisherige Reihenfolge: E
  - Rufe den Algorithmus rekursiv für die Nachbarn von E auf: B, D, G, I



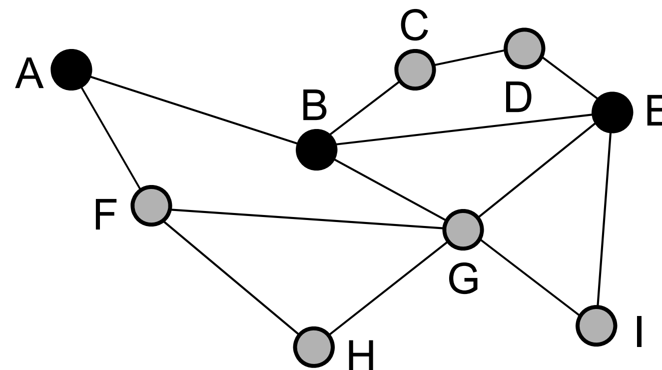
Beispiel (III)

- Durchgang 2: Rufe Prozedur für Knoten B auf
  - Markiere B
  - Gebe B aus, bisherige Reihenfolge: E B
  - Rufe den Algorithmus rekursiv für die Nachbarn von B auf: A, C, E, G



Beispiel (IV)

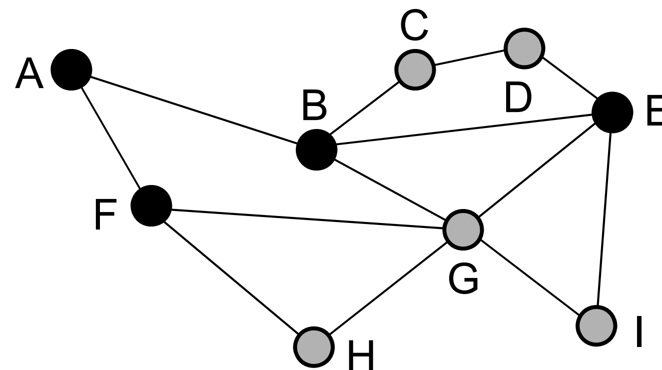
- Durchgang 3: Rufe Prozedur für Knoten A auf
  - Markiere A
  - Gebe A aus, bisherige Reihenfolge: E B A
  - Rufe den Algorithmus rekursiv für die Nachbarn von A auf: B, F



- Durchgang 4: Rufe Prozedur für Knoten B auf
  - B bereits markiert, keine Aktion im Prozedurrumpf

Beispiel (V)

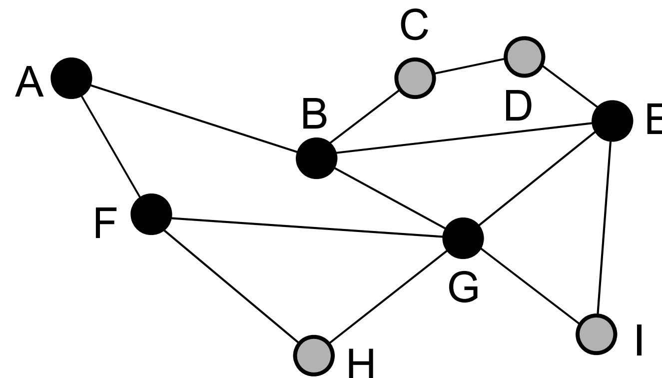
- Durchgang 5: Rufe Prozedur für Knoten F auf
  - Markiere F
  - Gebe F aus, bisherige Reihenfolge: E B A F
  - Rufe den Algorithmus rekursiv für die Nachbarn von F auf: A, G, H



- Durchgang 6: Rufe Prozedur für Knoten A auf
  - A bereits markiert, keine Aktion im Prozedurrumpf

Beispiel (VI)

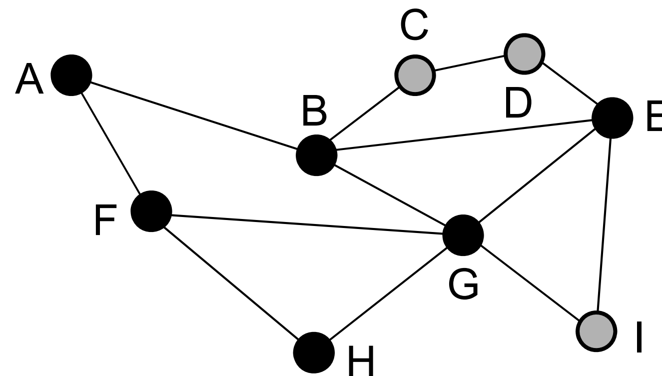
- Durchgang 7: Rufe Prozedur für Knoten G auf
  - Markiere G
  - Gebe G aus, bisherige Reihenfolge: E B A F G
  - Rufe den Algorithmus rekursiv für die Nachbarn von G auf: B, E, F, H, I



- Durchgang 8: Rufe Prozedur für Knoten B auf
  - B bereits markiert, keine Aktion im Prozedurrumpf
- Durchgang 9: Rufe Prozedur für Knoten E auf
  - E bereits markiert, keine Aktion im Prozedurrumpf
- Durchgang 10: Rufe Prozedur für Knoten F auf
  - F bereits markiert, keine Aktion im Prozedurrumpf

Beispiel (VII)

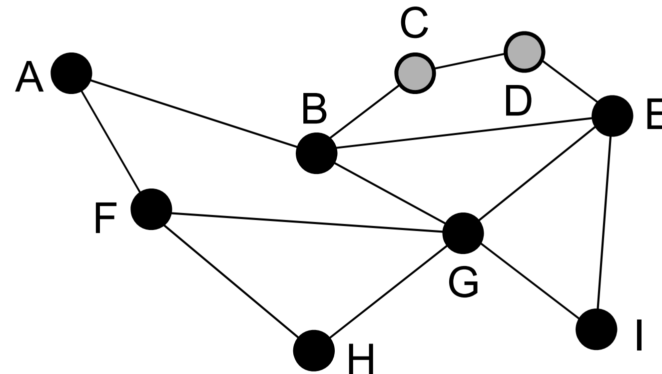
- Durchgang 11: Rufe Prozedur für Knoten H auf
  - Markiere H
  - Gebe H aus, bisherige Reihenfolge: E B A F G H
  - Rufe den Algorithmus rekursiv für die Nachbarn von H auf: F, G



- Durchgang 12: Rufe Prozedur für Knoten F auf
  - F bereits markiert, keine Aktion im Prozedurrumpf
- Durchgang 13: Rufe Prozedur für Knoten G auf
  - G bereits markiert, keine Aktion im Prozedurrumpf

Beispiel (VIII)

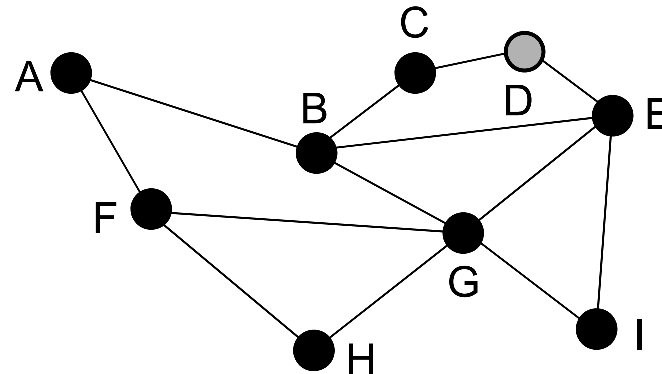
- Durchgang 14: Rufe Prozedur für Knoten I auf (aus Durchgang 7)
  - Markiere I
  - Gebe I aus, bisherige Reihenfolge: E B A F G H I
  - Rufe den Algorithmus rekursiv für die Nachbarn von I auf: E, G



- Durchgang 15: Rufe Prozedur für Knoten E auf
  - E bereits markiert, keine Aktion im Prozedurrumpf
- Durchgang 16: Rufe Prozedur für Knoten G auf
  - G bereits markiert, keine Aktion im Prozedurrumpf
- Durchgang 17: Rufe Prozedur für Knoten H auf (aus Durchgang 5)
  - H bereits markiert, keine Aktion im Prozedurrumpf

Beispiel (IX)

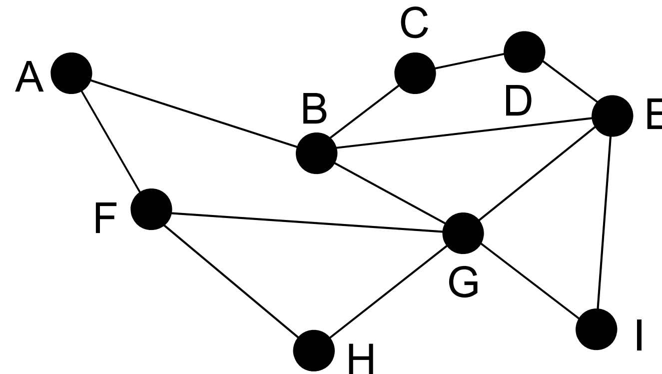
- Durchgang 18: Rufe Prozedur für Knoten C auf (aus Durchgang 2)
  - Markiere C
  - Gebe C aus, bisherige Reihenfolge: E B A F G H I C
  - Rufe den Algorithmus rekursiv für die Nachbarn von C auf: B, D



- Durchgang 19: Rufe Prozedur für Knoten B auf
  - B bereits markiert, keine Aktion im Prozedurrumpf

Beispiel (X)

- Durchgang 20: Rufe Prozedur für Knoten D auf
  - Markiere D
  - Gebe D aus, bisherige Reihenfolge: E B A F G H I C D
  - Rufe den Algorithmus rekursiv für die Nachbarn von D auf: C, E



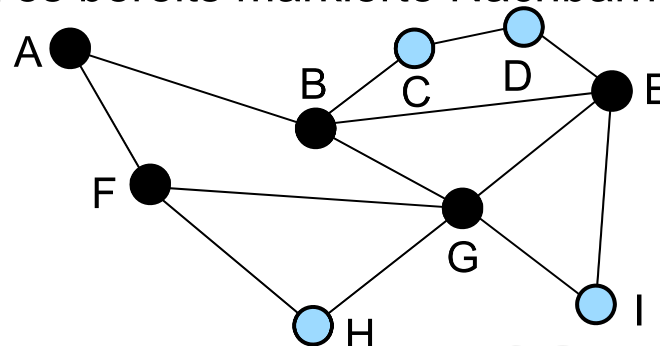
- Durchgang 21ff: Rekursion baut sich ab

Komplexität

- Komplexität: O(n+m)
  - n Schritte sind notwendig, um alle Knoten als nicht markiert zu initialisieren
  - Maßgebend ist Anzahl rekursiver Aufrufe
  - Entspricht 2·m (Jede Kante wird von jedem Knoten aus aufgerufen)

Anwendung (I): Zyklenfreiheit

- Häufiges Problem: Zyklenfreiheit eines Graphen muss nachgewiesen werden
- Beispiel:
  - Knoten sind zu erledigende Aufgaben
  - Kante von A nach B existiert, falls A vor B erledigt werden muss
  - Nur zyklenfreier Graph gibt konsistente Definitionen wieder
- Test auf Zyklenfreiheit kann mittels Tiefensuche gelöst werden
- Bedingung hierbei: Ist beim rekursiven Aufruf ein Nachbar (außer dem gerade zuvor besuchten) bereits markiert, dann liegt ein Zyklus vor
- Im Beispiel ist dies in Durchgang 7 der Fall:
  - G ist der aktuelle Knoten
  - Mit B und E gibt es bereits markierte Nachbarn



Anwendung (II): Zusammenhang (I)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Modifikation der Tiefensuche
- Idee:
  - Iteration über alle Knoten
  - Finde von hier alle erreichbaren Knoten
  - Vergebe pro Zusammenhangskomponente eine Nummer
  - Knoten werden mit dieser Nummer markiert (Feld `ZKNummer`)
  - Knoten mit gleicher Nummer bilden Zusammenhangskomponente

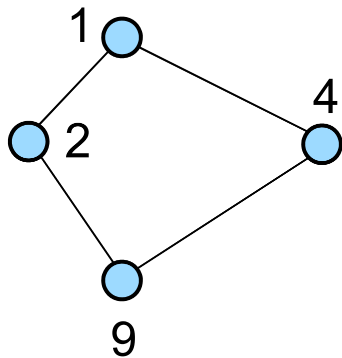
Anwendung (II): Zusammenhang (II)

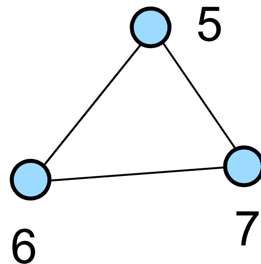

```
( 1) // Initialisierungen
( 2) for each v ∈ V {
( 3)   ZKNumber[v] = 0;
( 4) }
( 5) counter = 0;
( 6) // Iteration über alle Knoten
( 7) for each v ∈ V {
( 8)   if (ZKNumber[v] == 0){
( 9)     counter++;
(10)     connectivity(v);
(11)   }
(12) }
```

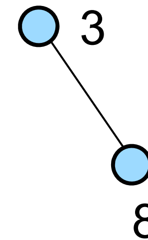
```
( 1) void connectivity(v){
( 2)   ZKNumber[v] = counter
( 3)   for each w ∈ neighbours(v){
( 4)     if (ZKNumber[w] == 0){
( 5)       connectivity(w);
( 6)     }
( 7)    }
( 8) }
```

Beispiel (I)

- Finde Zusammenhangskomponenten für Graphen G:







- Initialisierung:

Knoten	1	2	3	4	5	6	7	8	9
ZKNumber	0	0	0	0	0	0	0	0	0

```
counter = 0
```

Beispiel (II)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Iteration für Knoten 1:
  - Bedingung Zeile 8 erfüllt, `counter` wird inkrementiert, `connectivity` aufgerufen, alle Knoten der Komponente mit 1 markiert

Knoten	1	2	3	4	5	6	7	8	9
ZKNumber	1	1	0	1	0	0	0	0	1

`counter = 1`

- Iteration für Knoten 2:
  - Bedingung Zeile 8 nicht erfüllt, keine Aktion
- Iteration für Knoten 3:
  - Bedingung Zeile 8 erfüllt, `counter` wird inkrementiert, `connectivity` aufgerufen, alle Knoten der Komponente mit 2 markiert

Knoten	1	2	3	4	5	6	7	8	9
ZKNumber	1	1	2	1	0	0	0	2	1

`counter = 2`

Beispiel (III)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

- Iteration für Knoten 4:
  - Bedingung Zeile 8 nicht erfüllt, keine Aktion
- Iteration für Knoten 5:
  - Bedingung Zeile 8 erfüllt, `counter` wird inkrementiert, `connectivity` aufgerufen, alle Knoten der Komponente mit 3 markiert

Knoten	1	2	3	4	5	6	7	8	9
ZKNumber	1	1	2	1	3	3	3	2	1

```
counter = 3
```

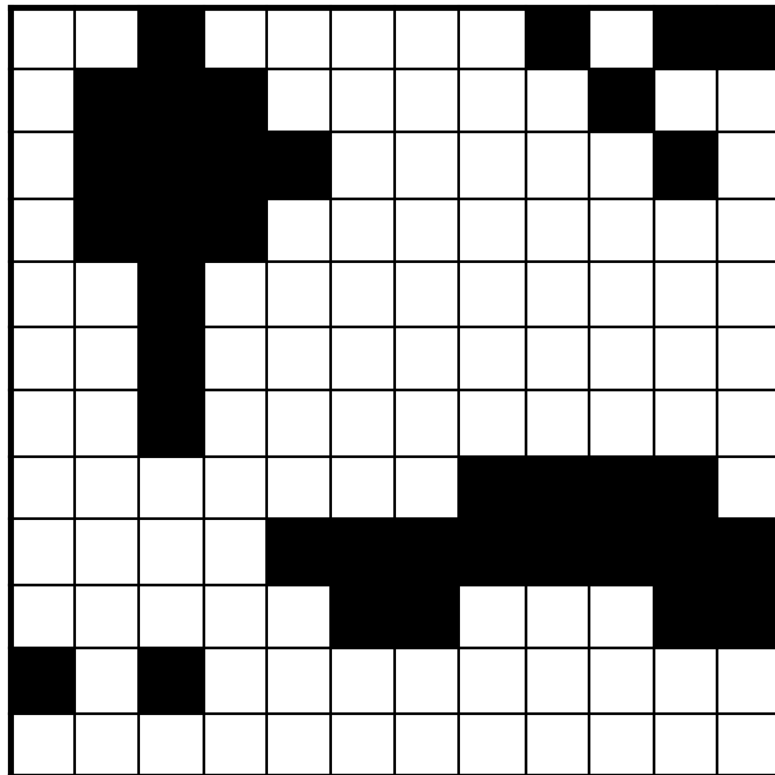
- Iterationen für Knoten 6 bis 9:
  - Bedingung Zeile 8 jeweils nicht erfüllt, keine Aktion

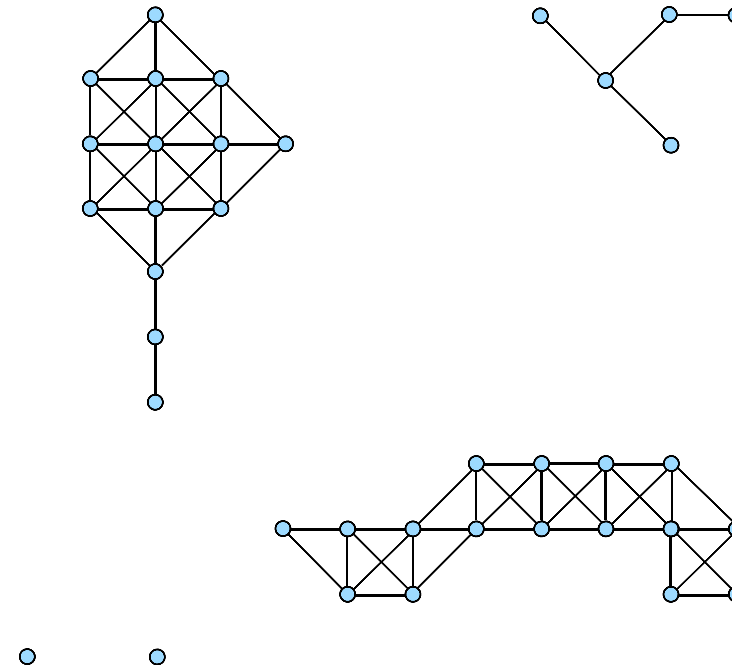
Anwendung (III): Bildverstehen (I)

- Wesentliches Teilgebiet der Künstlichen Intelligenz (KI)
- Einzelne Bildpunkte müssen zu Gebilden zusammengefasst und als Objekte erkannt werden
- Ausgangspunkt: Digitalisiertes Bild (d.h. Grauwerte)
- Überführung in binäres (schwarz-weißes) Bild
- Bildpunkte mit Wert 1 sind Objekte, die anderen Hintergrund
- Definition folgender Graph (bez. als Nachbarschaftsgraph):
  - Bildpunkte mit Wert 1 sind Knoten
  - Kante zwischen zwei Bildpunkten, wenn diese benachbart
  - Nachbarschaft:
    - 8-Zusammenhang: Berücksichtigung aller 8 Nachbarn
    - 4-Zusammenhang: Berücksichtigung nur Nachbarn oben, unten, links und rechts
- Graph analysieren:
  - Zusammenhangskomponenten Teile des Bildes
  - Zu kleine Komponenten können Rauschen sein

Anwendung (III): Bildverstehen (II)

- Beispiel: Binäres Bild und zugehöriger Nachbarschaftsgraph





[Tura04]

- Tiefensuche kann unmittelbar auf binäre Bildmatrix angewendet werden

Zusammenfassung (I)

- Transitiver Abschluss:
  - Definition
  - Adjazenzmatrizenmultiplikationsverfahren
  - Warshall-Algorithmus
  - Anwendungsbeispiel

- Minimal aufspannende Bäume:
  - Definition
  - Algorithmus von Prim
  - Algorithmus von Kruskal
  - Anwendungsbeispiele

Zusammenfassung (II)

- Suchverfahren in Graphen:
  - Tiefensuche ↔ Breitensuche

- Breitensuche:
  - Algorithmus mit Schlange
  - Anwendungsbeispiele

- Tiefensuche:
  - Algorithmus mit Stapel
  - Rekursiver Algorithmus
  - Zyklenfreiheit
  - Zusammenhangskomponenten
  - Anwendungsbeispiele

Übungen (I)

- **Aufgabe 1**

  Finden Sie im folgenden Rätsel fünf Begriffe aus diesem Vorlesungsabschnitt!

```
V P T V A Q F E U G X Z C B G C Y S C W B O M M L
O P Z R F K V G U T O X R N W W G W C M T Q H I R
E V J M A S H C N L B E E I Z N Q T P J I Z D N E
V E U U L N E P L P I M K V K S E J K X I C U I H
P H R N S U S O R T Y J K X H G J I G D N R X M H
D Y X X Q T C I E H C U S N E F E I T V P A E A I
E C O J Q C S N T X F D O O B M D M U F O N V L S
I Q L C Q N S S D I K U P Z J S U W O Y W O Z E Q
E P G Y D U G V O N V U K N Z U M H V Y I F Z R Y
B X I U C C G E D U H E Z B F T N Y M N X U Y S L
L O Z H N Z Y V B I I T R K F Y I T A H H D K P Y
F V E I N T W L R A Z M U A H N A P S Z Q B R A J
Z R A E M V T J Y R Z U Q V B Y S P V M R D K N S
M M G U R Y Z E Y E K Y C P T S L C E Q H I M N W
H H Y S B G L H N O V I U T H C C M H I N W G B J
W R N P F Y D W M D S A S O J K K H Z X S H O A X
S F X O Y M P F V E S U T V S V V P L S C I G U S
Q D P J Q O L U U G H S X O R X S B M U V B E M X
K A K D I R K R J T P V Y Y H I A X G F S M S M P
E R R E I C H B A R K E I T S M A T R I X S J A U
```

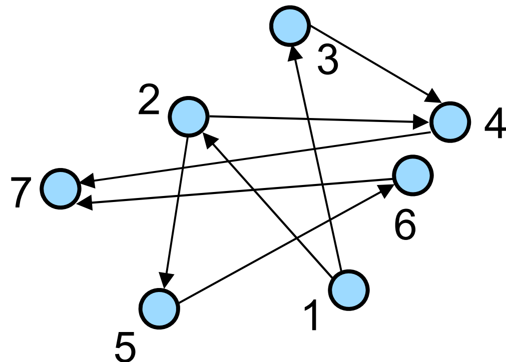
Übungen (II)

- **Aufgabe 2**
  a) Was ist der transitive Abschluss eines Graphen?
  b) Was ist die Erreichbarkeitsmatrix?
  c) Was berechnet der Warshall-Algorithmus und wie funktioniert er?
  d) Was ist ein (minimaler) Spannbaum?
  e) Welcher Algorithmus berechnet einen minimalen Spannbaum?
  f) Welche Suchverfahren in Graphen gibt es?
  g) Wie funktioniert Breitensuche, wie Tiefensuche?
  h) Nennen Sie Anwendungen der Breiten- und der Tiefensuche?

Übungen (III)

- **Aufgabe 3**

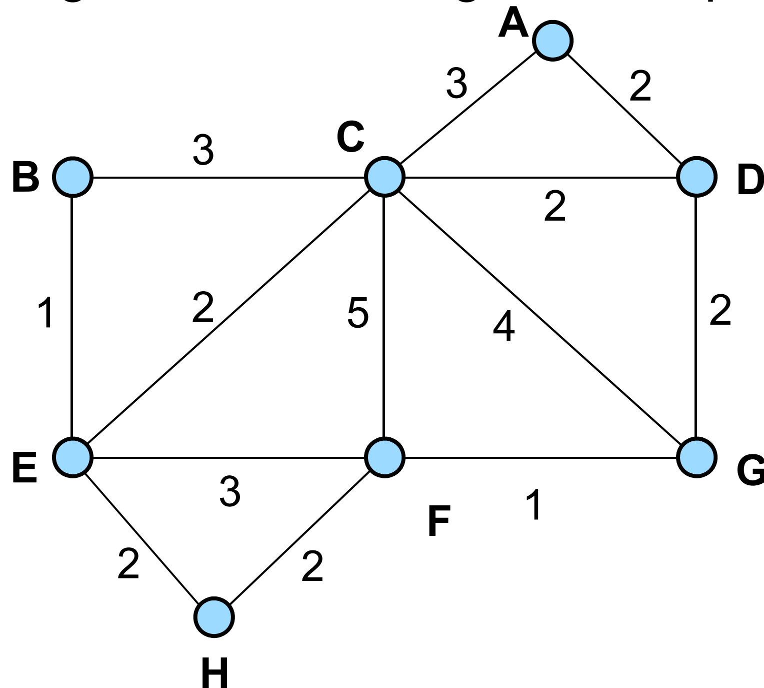
  Gegeben sei der folgende gerichtete Graph G.



a) Geben Sie den transitiven Abschluss von G visuell an!
b) Ermitteln Sie den transitiven Abschluss mit Hilfe der beiden in der VL vorgestellten Verfahren!
c) Wie ist der Aus- bzw. Eingangsgrad von Knoten 4?
d) Ist G regulär? Warum (nicht)?
e) Ist G azyklisch?
f) Ist G planar?

Übungen (IV)

Gegeben sei der folgende Graph G:



- **Aufgabe 4**

  Ermitteln Sie für G durch Anwendung des Algorithmus von Prim drei verschiedene minimal aufspannende Bäume!

- **Aufgabe 5**

  Ermitteln Sie für G durch Anwendung des Algorithmus von Kruskal einen minimal aufspannenden Baum!

Übungen (V)

- **Aufgabe 6**

  Der folgende rekursive Algorithmus A konstruiert einen aufspannenden Baum:

  Gegeben: Ungerichteter Graph G = (V,E)

  (1) Markiere einem beliebigen Knoten $v \in V$

  (2) Wiederhole für alle von v ausgehenden Kanten $(v,v') \in E$:

  - Wenn v’ unmarkiert, dann markiere v’ und rufe A rekursiv für v’ auf
  - Sonst: Lösche Kante (v,v’), wenn v’ nicht die Kante ist, von der aus v markiert worden ist

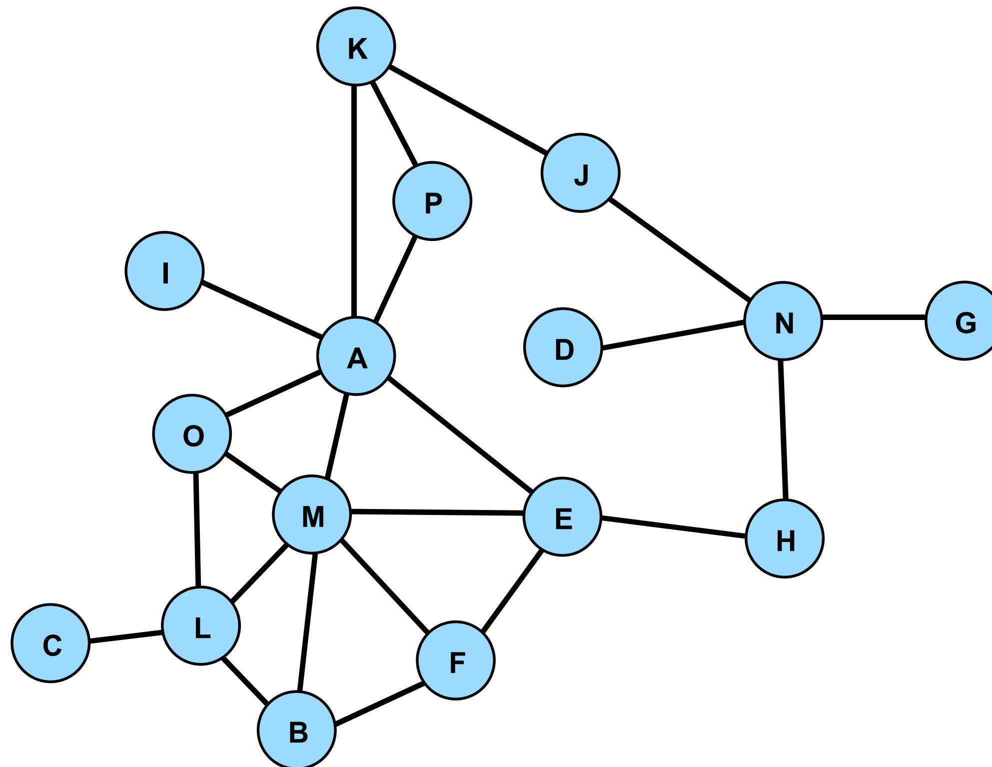
  a) Welche Komplexität hat dieser Algorithmus?

  b) Demonstrieren Sie die Arbeitsweise des Algorithmus am Beispiel des Graphen aus Aufgabe 4!

  c) Ist der ermittelte Spannbaum minimal?

Übungen (VI)

- **Aufgabe 7**
  Gegeben sei der folgende Graph:



Geben Sie das Resultat eines Breiten- und eines Tiefendurchlaufs, beginnend bei Knoten A, an! Geben Sie auch die einzelnen Schritte des jeweiligen Algorithmus an!

Übungen (VII)

- **Aufgabe 8**

  Wenden Sie auf folgenden Graphen den Algorithmus zum Auffinden von Zusammenhangskomponenten an: