DHBW Lörrach
Duale Hochschule Baden-Württemberg

Algorithmen und Komplexität
TIF 21 A/B
Dr. Bruno Becker

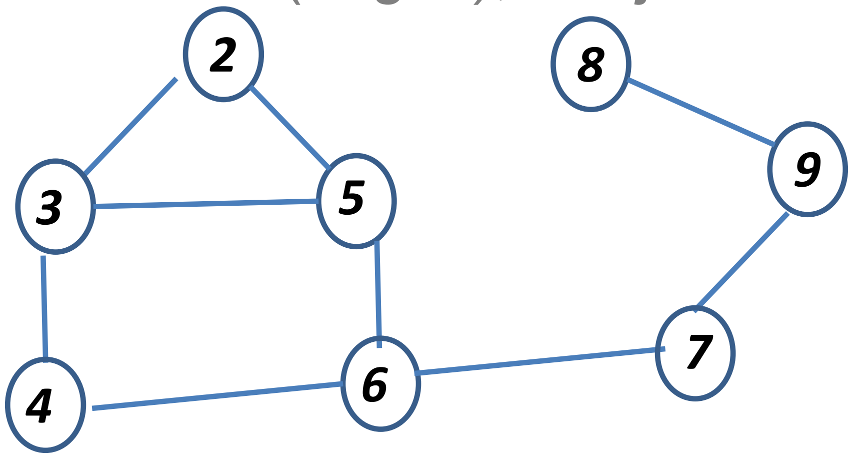
8. Graphenalgorithmen

Graphenalgorithmen

- **Definitionen und Begriffe**
- Datenstrukturen für Graphen
- Tiefensuche
- Breitensuche

Graphen

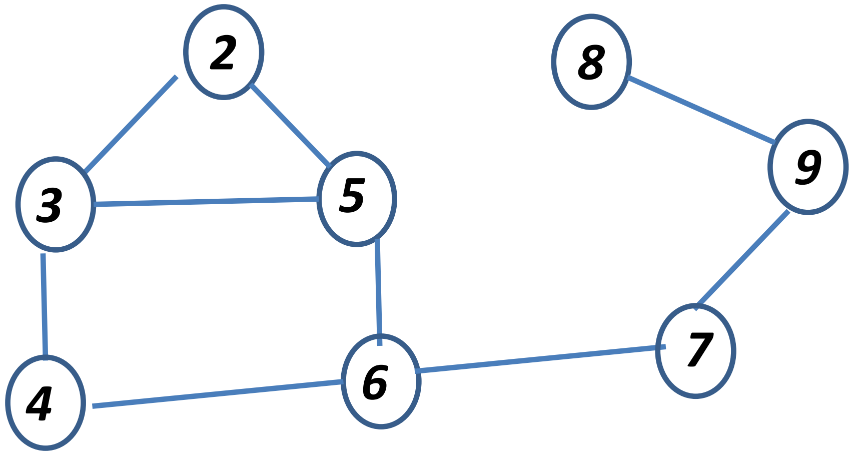
- Ein (**ungerichteter**) **Graph** ***G= (V,E)*** besteht aus
  - einer endlichen Menge *V* von **Knoten** (*vertices*)
  - einer Menge *E* von **Kanten** (*edges*), die je 2 Knoten verbinden



Ungerichteter Graph: Kanten sind bidirektional
Gerichteter Graph: Kanten sind *Pfeile* mit *Anfangs-* und *Endknoten*

Graphen: Begriffe

- **Benachbarte (adjazente) Knoten**: sind durch eine Kante verbunden
- **Grad** eines Knotens = Anzahl Nachbarn = Anzahl Kanten am Knoten
- **Pfad (Weg):** Folge durch Kanten verbundener Knoten
  - Muss nicht eindeutig sein
- **Länge** eines Pfades = Anzahl Kanten
- **Zyklus:** Pfad mit erstem = letztem Knoten

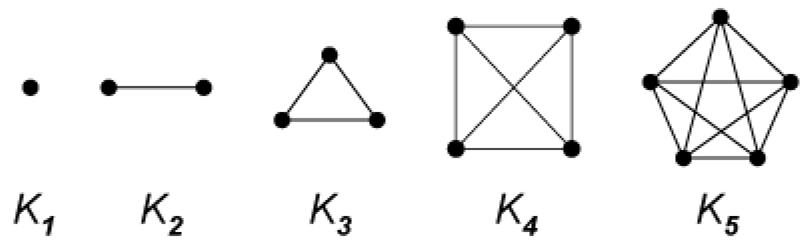


Anwendungen von Graphen

- Straßen- und Leitungsnetze
- Flugstrecken
- Computernetzwerke
- Soziale Netzwerke
- Situationen und Züge in Spielen
- Abhängigkeiten zwischen Aktivitäten in Projektplänen
- …
- Was sind hier jeweils die Knoten/Kanten?

Übung: Anzahl der Kanten in einem Graphen

- Graph mit 1,2, 3,4,5 Knoten und *maximaler* Zahl grafisch darstellen



- Wie ist allgemein die maximale Kantenanzahl für Graph mit *n* Kanten?

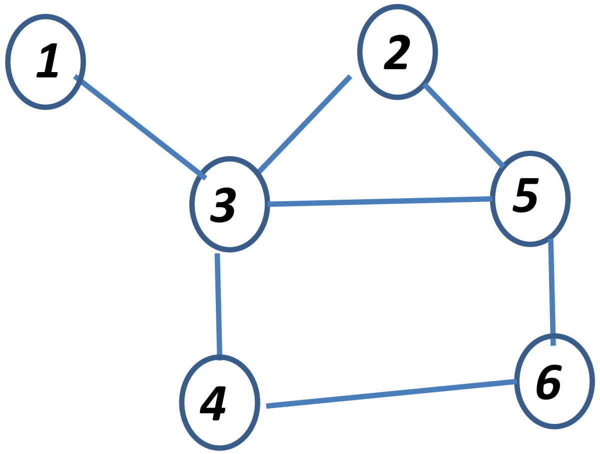
n*(n-1)/2

Graphenalgorithmen

- Definitionen und Begriffe
- **Datenstrukturen für Graphen**
- Tiefensuche
- Breitensuche

Datenstrukturen für Graphen

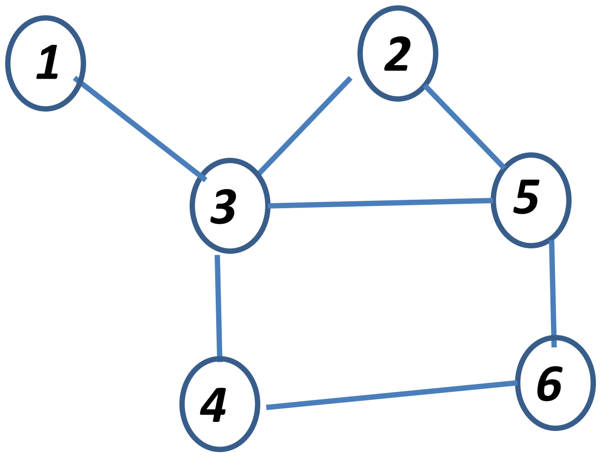
- **Abspeicherungen der Kanten**
  - in 2-dimensionalem Array -> Adjazenzmatrix
  - für jeden Knoten eine verkettete Liste mit Nachbarn -> Adjazenzliste



- Codierung des Graphen in Textdatei einfach möglich
  - 1-3, 2-3, 2-5, 3-4, 3-5, 4-6, 5-6

Adjazenzmatrix

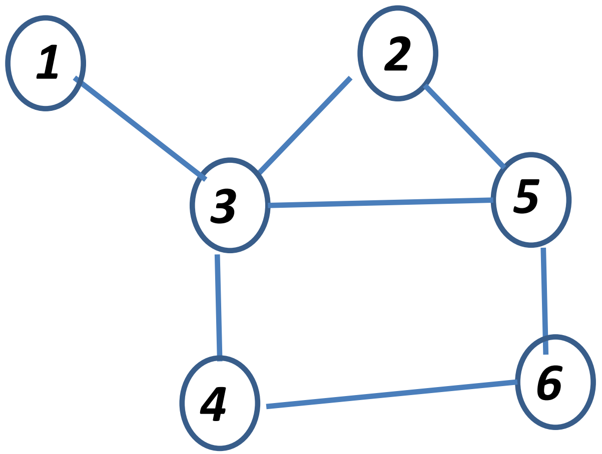
- **Matrix -** $\| V \| \times \| V \|$
  - $a_{ij} = 1$, falls es Kante zwischen *i* und *j* gibt, 0 sonst
  - Für ungerichtete Graphen reicht halbe Matrix (oberhalb oder unterhalb Diagonale) aus

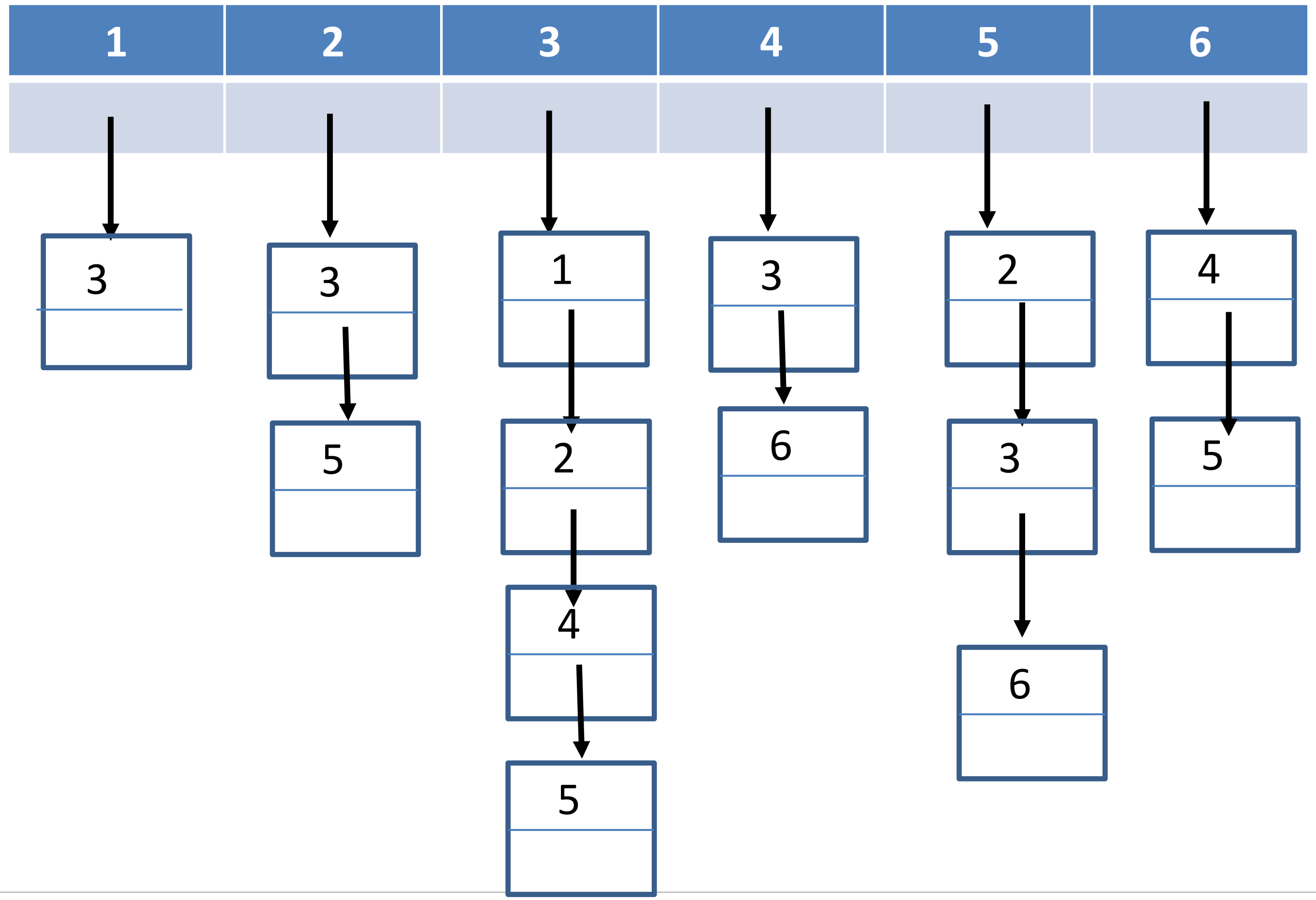


	1	2	3	4	5	6
1	0	0	1	0	0	0
2	0	0	1	0	1	0
3	1	1	0	1	1	0
4	0	0	1	0	0	1
5	0	1	1	0	0	1
6	0	0	0	1	1	0

Adjazenzliste

- **Array - $\| V \|$**
  - a_i enthält lineare Liste für die Nachbarn von *i*





Aufwandsbetrachtung

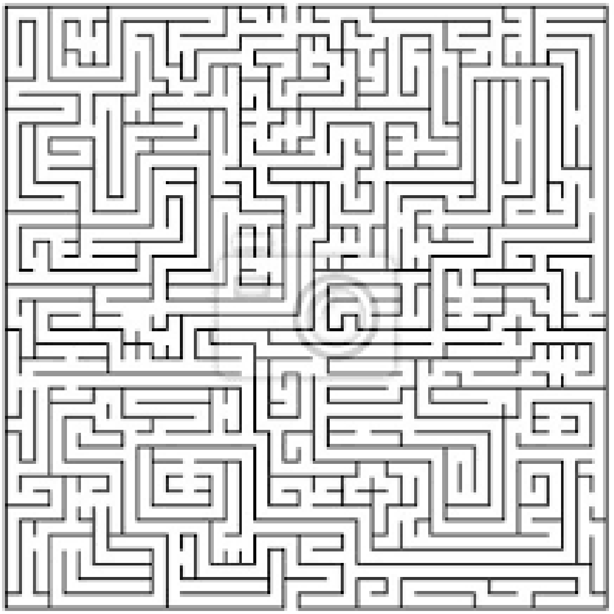
- **Adjazenzmatrix**
  - Statische Struktur
  - Bei n Knoten n^2 Speicherplatz
  - Matrix häufig sehr dünn besetzt -> viel ungenutzter Speicherplatz
- **Adjazenzliste**
  - „Halbdynamische" Struktur
  - Bei n Knoten und m Kanten $\Theta\,(n + m)$ Speicherplatz
  - Für viele Operationen (z.B. Verfolgen von Kanten) gut geeignet
  - „Volldynamische Variante"
    - Knoten in doppelt verketteter Liste mit Zeigern auf Vorgänger, Nachfolger, Kantenliste
    - Jeder Eintrag in Kantenliste enthält Zeiger auf Vorgänger, Nachfolger und Knoten

Graphenalgorithmen

- Definitionen und Begriffe
- Datenstrukturen für Graphen
- **Tiefensuche**
- Breitensuche

Suche in Graphen

- **Suchproblem (Traversierung)**
  - Gegeben Graph *G(V,E)* und hieraus Knoten *v* als Startpunkt
  - Finde *alle* von *v aus* erreichbaren Knoten und je einen Pfad dorthin
  - Erweiterung: Finde jeweils den *kürzesten* Pfad



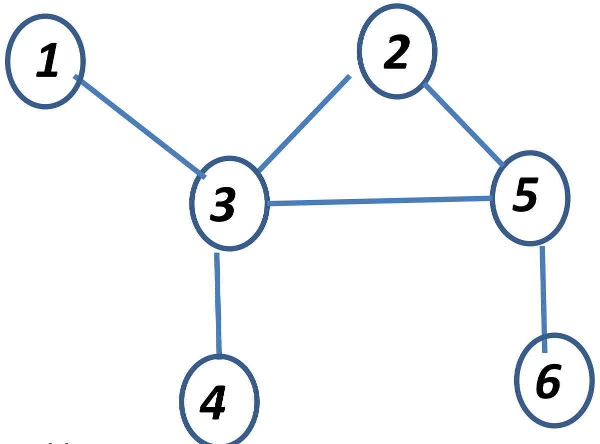
Klassisches Vorbild: Theseus im Labyrinth
Bindfaden als Hilfe

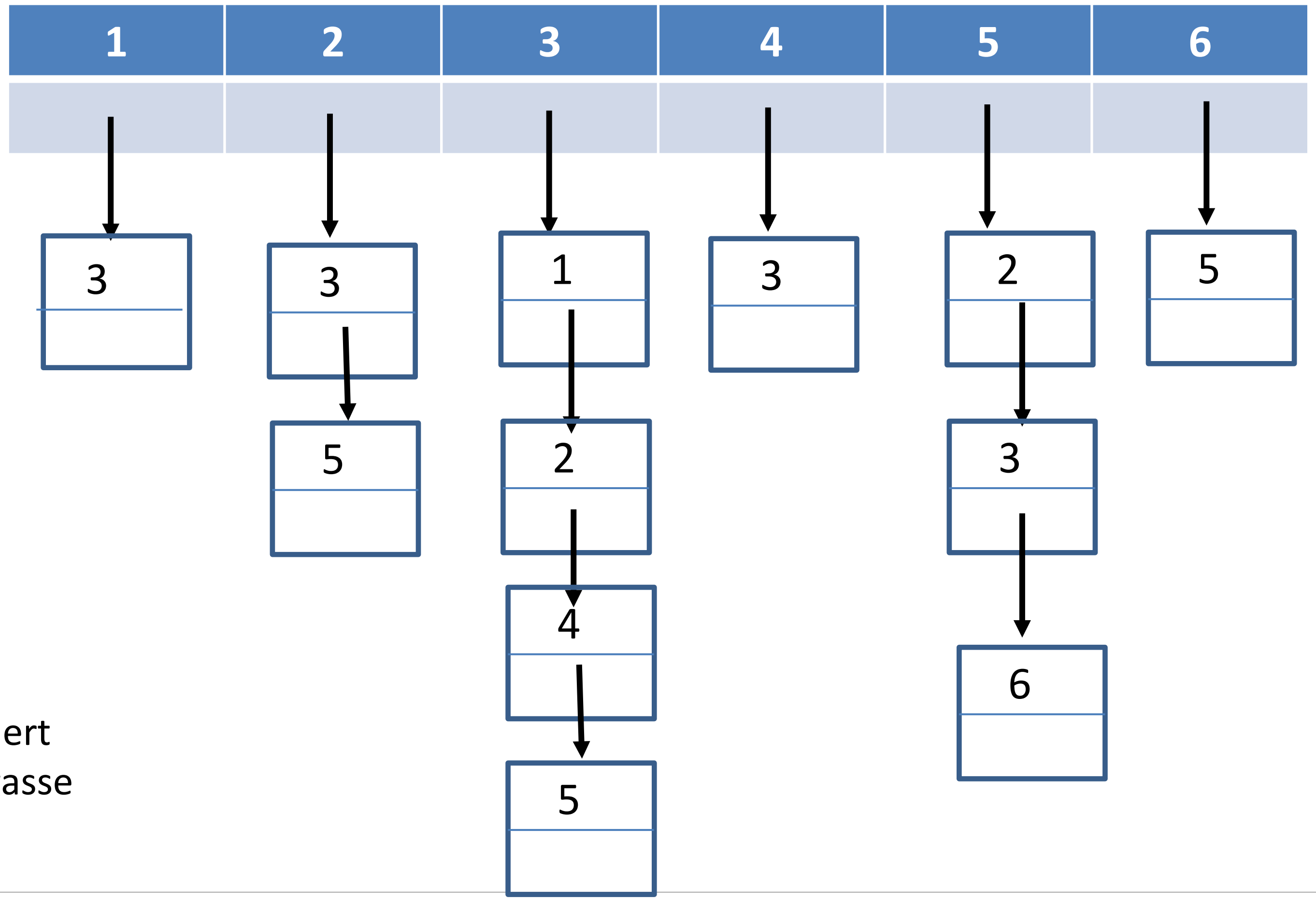
Tiefensuche (Depth-First-Search, DFS)

- **Rekursiver Algorithmus für Start-Knoten *v* aus Graph *G (V,E)***
  - Markiere *v* als besucht
  - Besuche rekursiv alle unmarkierten Nachbarn *w* von *v* (und speichere *v* als deren unmittelbaren Nachfolger im Suchpfad, d.h. Kante v-w führt zu Markierung von w)

- **Reihenfolge der besuchten Knoten**
  - Läuft solange bis zum nächsten Knoten, bis es keinen unmarkierten Nachbarn mehr gibt (Sackgasse).
  - Geht dann schrittweise zurück, bis wieder unmarkierter Nachbar vorhanden ist, bzw. alle Knoten markiert (Backtracking)
  - Reihenfolge innerhalb der Nachbarn so, wie in der Implementierung gegeben – z.B. sequentieller Durchlauf durch Adjazenzliste

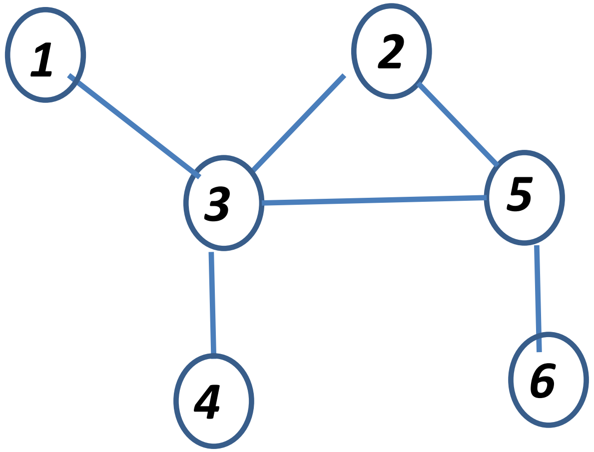
Beispiel Tiefensuche (DFS)

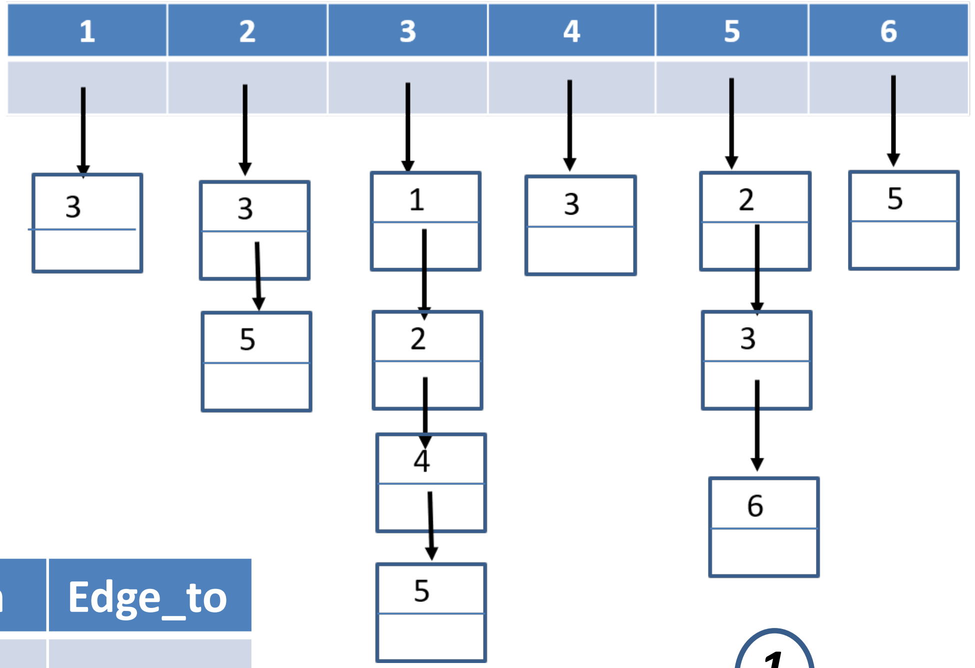




DFS(1) ; //1 Startknoten
Markiere 1;
DFS(3); //1. Nachbar (Nb) von 1
Markiere 3;
DFS(2); //1. unmarkierter Nb von 3
Markiere 2;
DFS(5); //1. unmarkierter Nb von 2
Markiere 5;
DFS(6) //Nb 2 und 3 sind markiert
Markiere 6; //Sackgasse
Fertig 6, 5, 2,
Markiere 4; //unmarkierter NB von 3

Beispiel DFS

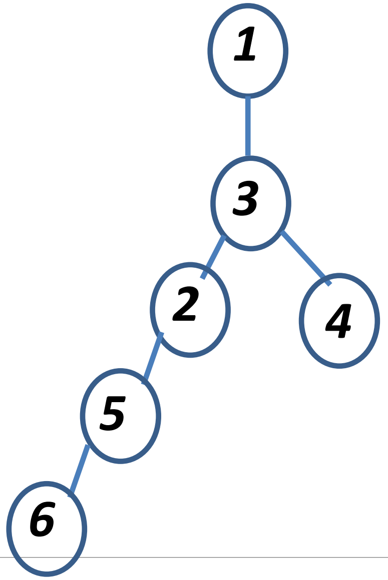




DFS(1) ; //1 Startknoten
Markiere 1;

DFS(3); //1. Nachbar (Nb) von 1
Markiere 3 **und merke Kante (1,3)**;
DFS(2); //1. unmarkierter Nb von 3
Markiere 2 **und merke Kante (3,2)**;
DFS(5); //1. unmarkierter Nb von 2
Markiere 5 **und merke Kante (2,5);**
DFS(6) //Nb 2 und 3 sind markiert
Markiere 6 **und Kante (5,6)**
Fertig 6, 5, 2,
Markiere 4 **und merke Kante (3,4)**

Knoten	Edge_to
1	--
2	**3**
3	**1**
4	**3**
5	**2**
6	**5**



Eigenschaften Tiefensuche

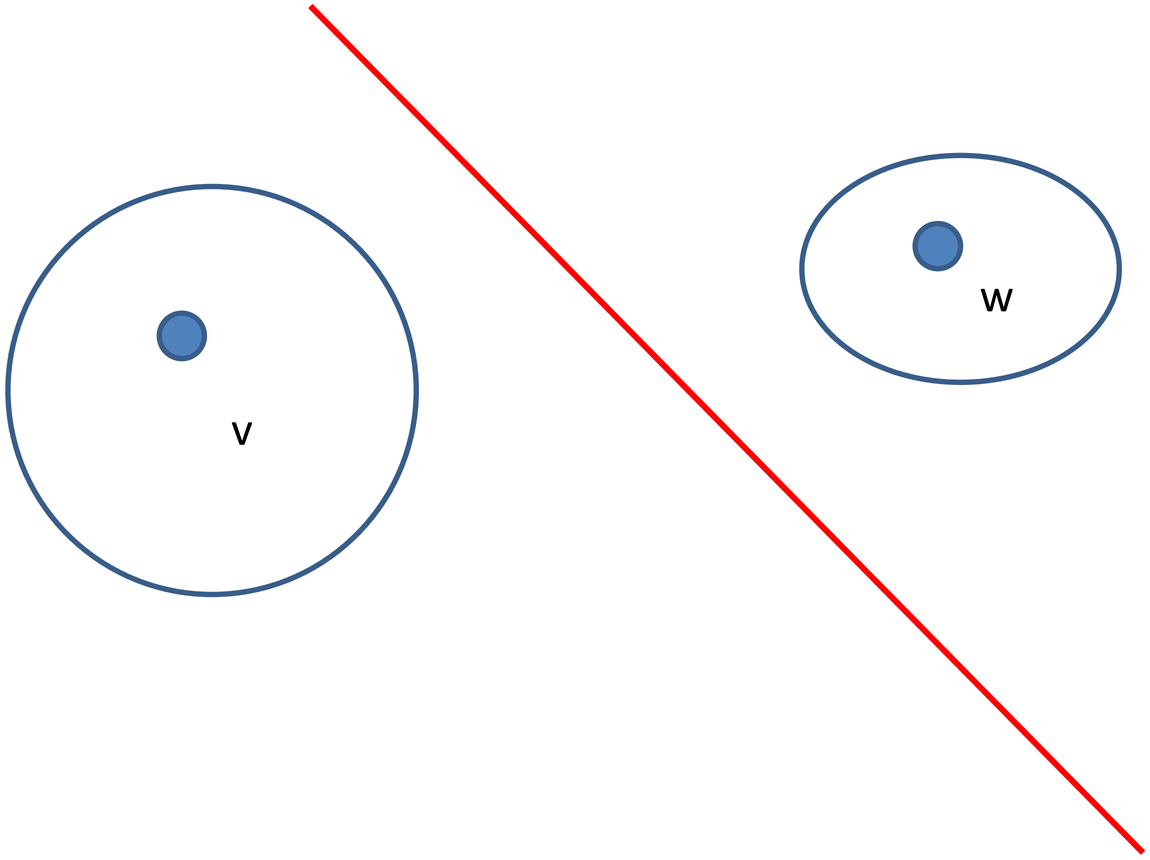
- **Aufwand**
  - Tiefensuche markiert alle Knoten, die mit Startknoten s verbunden sind, in einer Zeit proportional zur Summe der Grade
  - O($\| V \| + \| E \|$)
- **Anwendungen**
  - Gibt es zu einem Startknoten *s und* Zielknoten *v* im Graph *G* einen Pfad?
  - Wenn ja, gib einen Pfad aus (*Traversieren im Baum mit Wurzel s zu v)*
  - Liefert nicht unbedingt den kürzesten Pfad (Beispiel 1-3-2-5-6 statt 1-3-5-6)
  - Gut für strukturelle Informationen wie z.B. Zusammenhang eines Graphen

Zusammenhangskomponenten

- **Definition**
  - Ein ungerichteter Graph *G* heißt **zusammenhängend,** wenn es von jedem Knoten *v* zu jedem anderen Knoten *w* mindestens einen Pfad gibt.
  - Ein maximal zusammenhängender Teilgraph eines ungerichteten Graphen *G* heißt **Zusammenhangskomponente** (*connected component)* von *G.*

- **Algorithmus zur Ermittlung der Zusammenhangskomponenten**
  - Tiefensuche für einen beliebigen Startknoten findet dessen Zusammenhangskomponente. Markiere die Knoten dieses Komponente mit 1
  - Tiefensuche für beliebigen unmarkierten Knoten, markieren Knoten mit 2
  - …
  - Bis kein unmarkierter Knoten mehr vorhanden

Zusammenhangskomponenten - Visualisierung



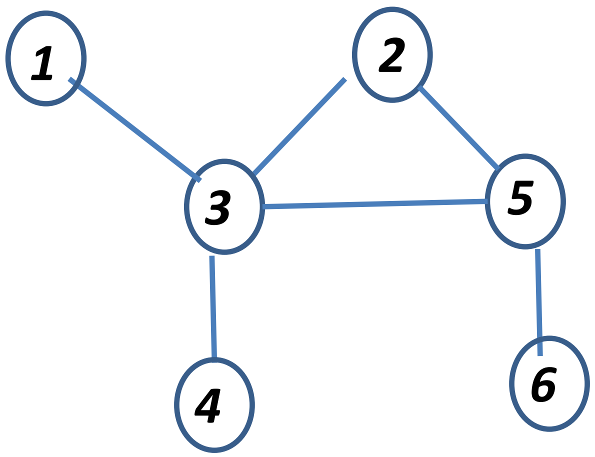
Graphenalgorithmen

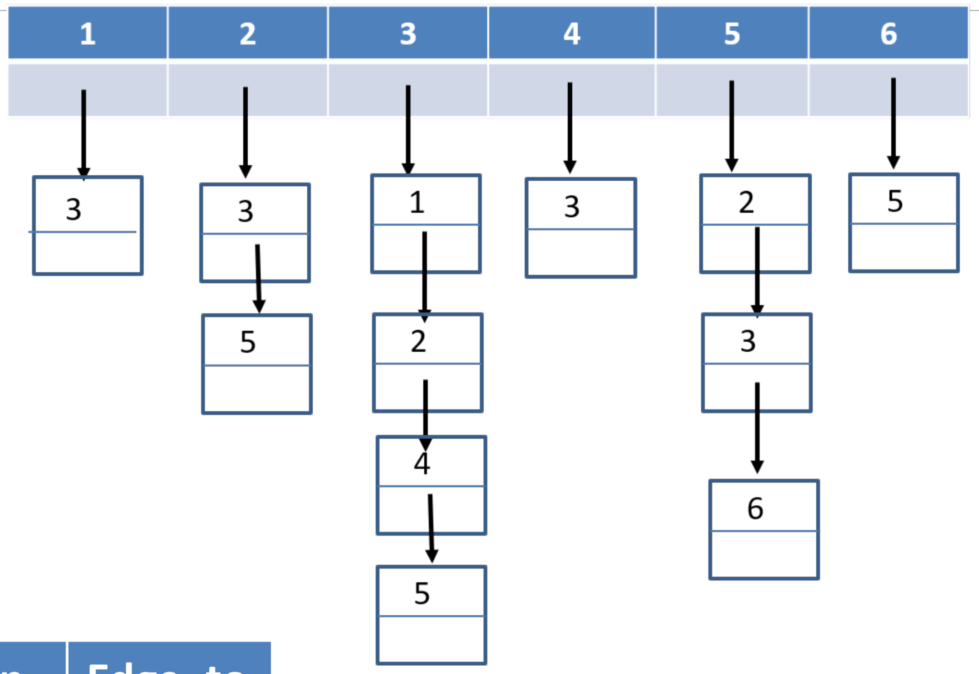
- Definitionen und Begriffe
- Datenstrukturen für Graphen
- Tiefensuche
- **Breitensuche**

Breitensuche (Breadth-First-Search, BFS)

- **Ziel und Lösungsansatz:**
  - Ziel: Kürzeste Pfade (d.h. kleinste Kantenzahl) vom Startknoten zu jedem erreichbaren Knoten finden
  - Idee: Knoten in aufsteigender Entfernung zum Start durchsuchen
  - Ansatz: Iterativer Algorithmus mit *Queue* (Warteschlange, FIFO)
- **Breitensuche**
  - Füge Startknoten *s* in Queue ein und markiere *s* als besucht
  - Wiederhole, bis Queue leer ist:
    - Entnimm den ältesten Knoten *v* aus der Queue
    - Füge alle unmarkierten Nachbarn von *v* in die Queue ein und markiere sie als besucht

Beispiel BFS



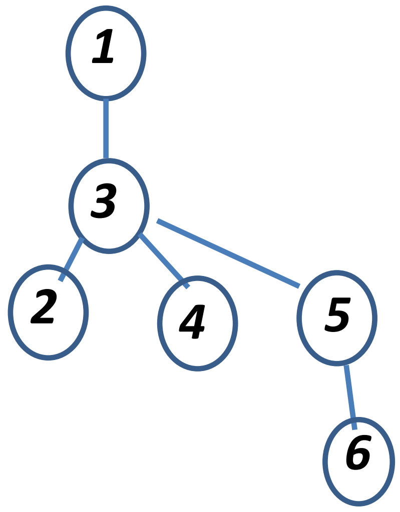


BFS(1) ; //1 Startknoten in Schlange (Q=(1))
Markiere 1;

1 aus Q , Nachbarn von 1 in Q= (3) ,
3 markieren und merke Kante (1,3);
3 aus Q, unmark. Nachbarn von 3 in Q =(2,4,5) und merke Kanten (3,2), (3,4) und (3,5);
2 aus Q, alle Nachbarn von 2 schon markiert Q= (3,5)
4 aus Q, alle Nachbarn von 4 schon markiert Q = (5)
5 aus Q, Nachbarn 6 in Q (6) und merke Kante (5,6) Q= (6)
6 aus Q, alle Nachbarn von 6 schon markiert Q = (0)

Fertig;

Knoten	Edge_to
1	--
2	**3**
3	**1**
4	**3**
5	**3**
6	**5**



Eigenschaften Breitensuche

- **Aufwand**
  - Zeit: O($\| V \| + \| E \|$), Platz O($\| V \|$)
- **Anwendungen**
  - Breitensuche berechnet für jeden Knoten *v,* der von *s* erreicht werden kann, einen kürzesten Pfad. (Beweis durch Induktion)
  - Das gilt für den Fall, dass Kanten alle gleich lang sind. In der Praxis haben Kanten unterschiedliches Gewicht, dann benötigt man anderes Verfahren für kürzesten Weg (-> Algorithmus von Dijkstra, später in der Vorlesung)

Zusammenfassung Tiefen- und Breitensuche

- **Aufgaben und Grenzen**
  - Tiefensuche gut für Strukturaufgaben (z.B. Zusammenhangskomponente)
  - Tiefensuche benötigt i.a. weniger Speicherplatz als Breitensuche, weil immer nur ein Pfad gemerkt werden muss
  - Tiefensuche „geht in die Knie“ wenn einzelne Pfade des Graphen extrem lang sind. Alternative z.B. Iterative Tiefensuche mit Tiefenbeschränkung
  - Breitensuche findet für ungewichtete Graphen alle kürzeste Wege
- **Animation der Algorithmen**
- https://www.cs.usfca.edu/~galles/visualization/DFS.html
- https://www.cs.usfca.edu/~galles/visualization/BFS.html