Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 10, Montag, 10. Januar 2011

(Graphen, breadth und depth first search)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Morgen, Dienstag, Besprechung des Ü8 (a,b-Bäume)
    - um 10.15 Uhr in Gebäude 51, SR 00-006, mit Ina
  - Ihre Erfahrungen mit dem Ü9 (Cache/IO-Effizienz)
- Graphen
  - Neben Feldern, Listen und Bäumen die häufigste Datenstruktur
    - Bäume sind eine spezielle Art von Graph
  - Darstellung im Rechner
  - Ein paar algorithmische Probleme auf Graphen
  - **Breitensuche** und **Tiefensuche** im Detail
  - **Übungsaufgabe (Ü10):** Berechnung der (Größe der) Zusammenhangskomponenten in zufälligen Graphen

Ihre Erfahrungen mit dem Ü9 (Cache)

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Die meisten fanden den theoretischen Teil (Aufgabe 1) schwieriger als den praktischen (Aufgabe 2)
  - "Auch wenn man 3 Wochen Zeit hat sollte man rechtzeitig anfangen"
  - Manche hat das Erstellen der Tabelle / Grafik viel Zeit gekostet
  - Die meisten fanden es vom Aufwand her angemessen

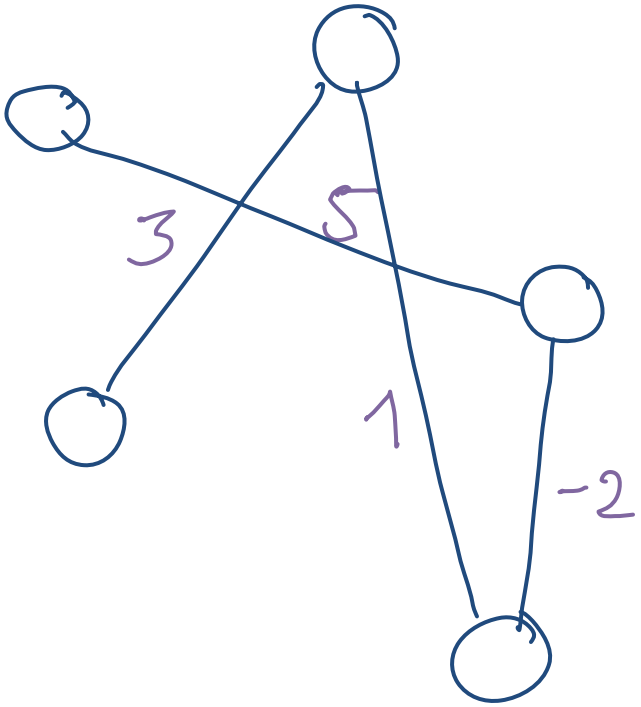
Graphen

UNI FREIBURG

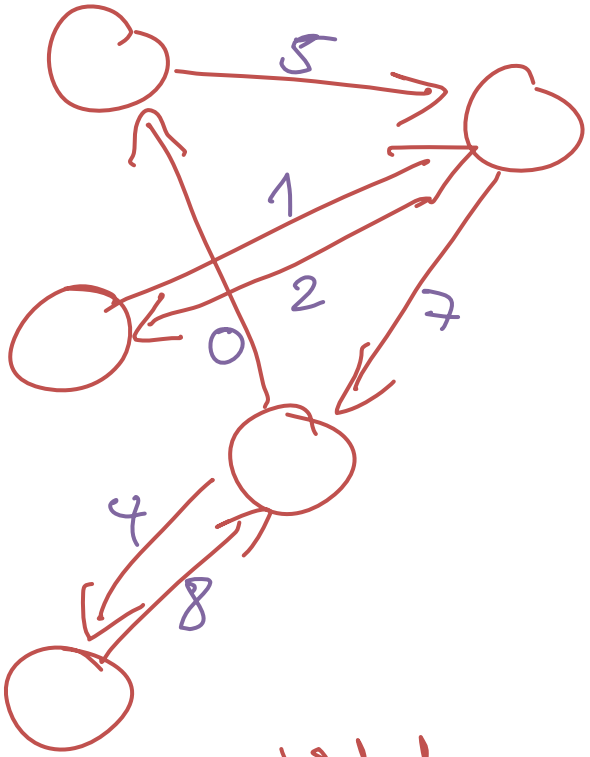
- Definition:
  - Ein Graph G besteht aus einer Menge V von Knoten …
    - Englisch: vertices (daher V) oder nodes
  - … und einer Menge E von Kanten
    - English: edges (daher E) oder arcs
  - Eine Kante e verbindet jeweils zwei Knoten u und v
    - ungerichtete Kante: $e = \{u, v\}$ (Menge)
    - gerichtete Kante: $e = (u, v)$ (Tupel)
  - Gewichteter Graph
    - eine reelle Zahl pro Kante, das sog. Kantengewicht, je nach Anwendung auch Länge der Kante oder Kosten der Kante genannt

Graphen

- Beispiele



ein ungerichteter Graph mit Kantengewichten

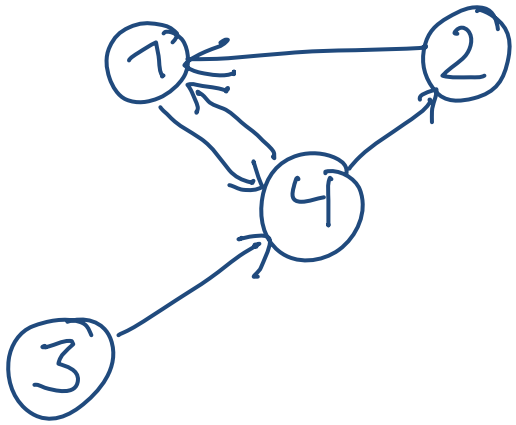


ein gerichteter Graph mit KGn

Repräsentation im Rechner

- Zwei klassische Arten
  - Adjazenzmatrix
    - Platzverbrauch ~ $|V|^2$
  - Adjazenzlisten bzw. -felder
    - Platzverbrauch ~ $|E|$
- Beispiel

	1	2	3	4
1	0	0	0	1
2	1	0	0	0
3	0	0	0	1
4	1	1	0	0



1 : 4
2 : 1
3 : 4
4 : 1, 2

ausgehenden Kanten

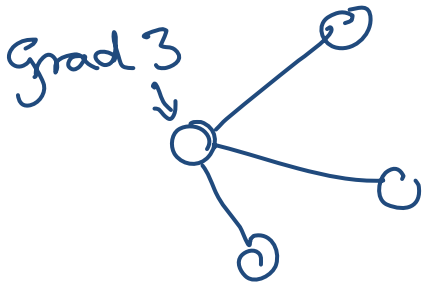
Grad

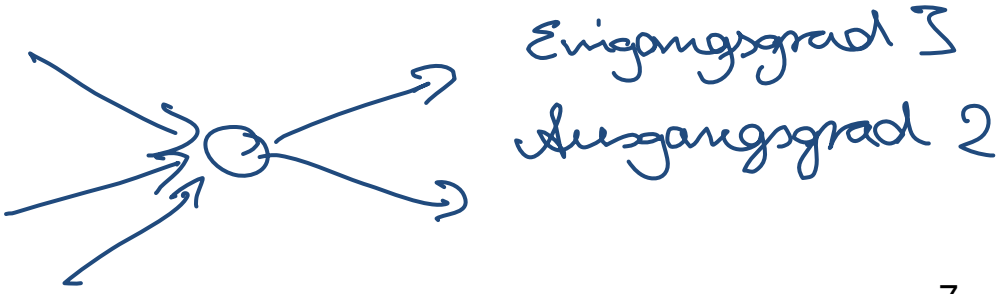
- Für einen Graphen $G = (V, E)$
  - falls gerichtet
    - Eingangsgrad von einem Knoten u

      = Anzahl eingehender Kanten = $|(v,u) : (v,u) \in E|$
    - Ausgangsgrad von einem Knoten u

      = Anzahl ausgehender Kanten = $|(u,v) : (u,v) \in E|$
  - falls ungerichtet
    - Grad von einem Knoten u

      = Anzahl adjazenter Kanten = $|\{u,v\} : \{u,v\} \in E|$



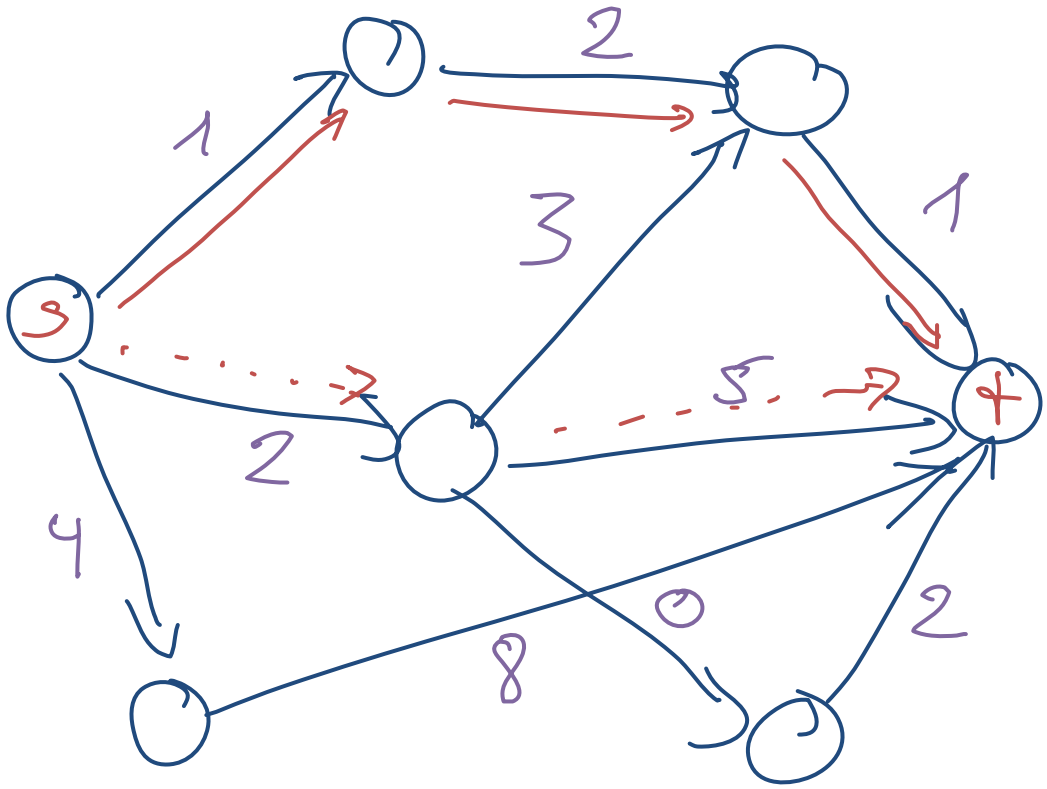


Pfade

- Für einen Graphen $G = (V, E)$
  - Ein Pfad in G ist eine Folge $u_1, u_2, u_3, \ldots, u_l \in V$ mit
    - $(u_1,u_2), (u_2,u_3), \ldots, (u_{l-1},u_l) \in E$ [gerichteter Graph]
    - $\{u_1,u_2\}, \{u_2,u_3\}, \ldots, \{u_{l-1},u_l\} \in E$ [ungerichteter Graph]
  - Falls die Kanten Gewichte habe, ist die Summe der Gewichte dieser Kanten das (Gesamt)-Gewicht des Pfades
    - bzw. die Länge des Pfades oder Kosten des Pfades
  - Der kürzeste Pfad (engl. shortest path) zwischen zwei Knoten u und v ist der Pfad $u, \ldots, v$ mit der kürzesten Länge
  - Der Durchmesser eines Graphen ist der längste kürzeste Pfad = $\max_{u,v}${Länge von P : P ist ein kürzester Pfad zwischen u und v}

Pfade

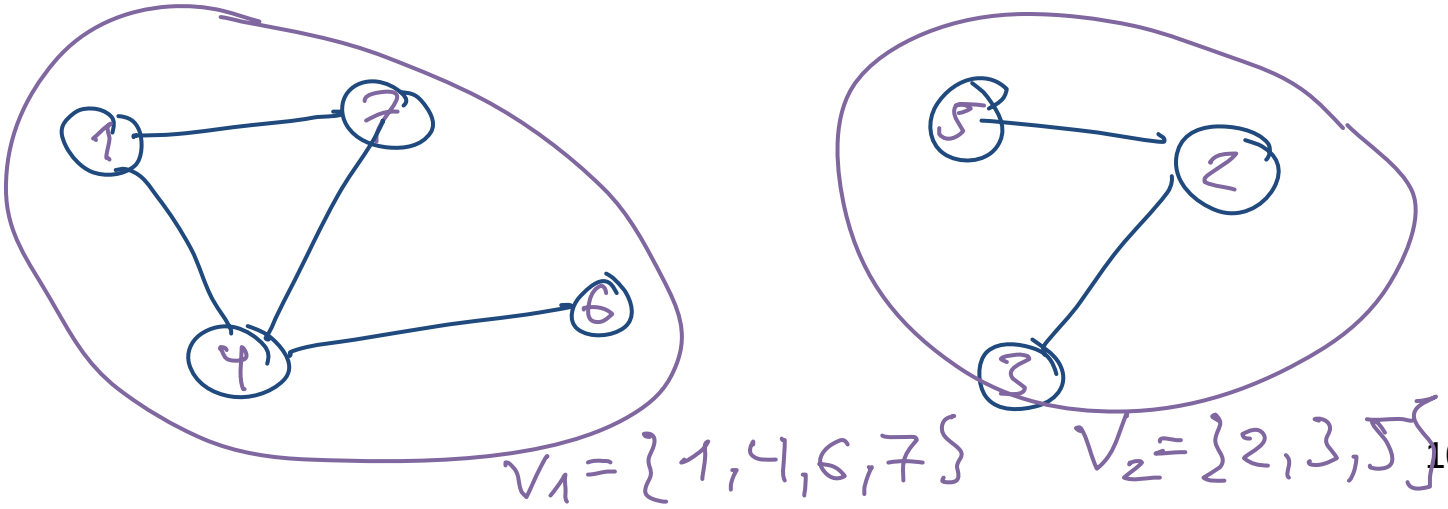
- Beispiel



Zusammenhangskomponenten

- Für einen ungerichteten Graphen $G = (V,E)$
  - Die Zusammenhangskomponenten bilden eine Partition von V, also $V = V_1 \cup \ldots \cup V_k$
  - Zwei Knoten u und v sind in derselben Zusammenhangskomponente, wenn es einen Pfad zwischen u und v gibt

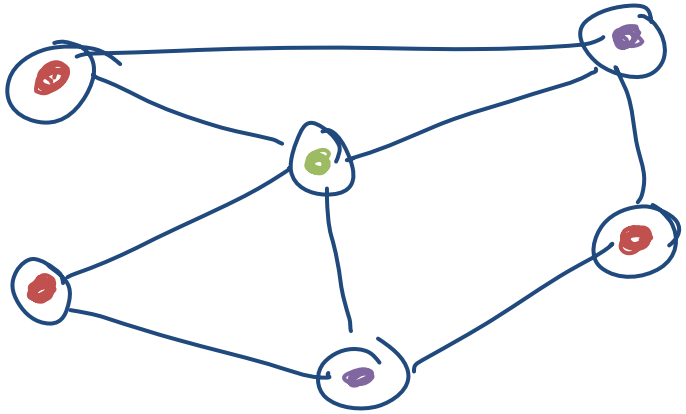
[für gerichtete Graphen ist die Definition komplizierter, man spricht dann von starken Zusammenhangskomponenten, das kommt vielleicht in einer späteren Vorlesung noch]



Graphenprobleme 1 / 5

UNI FREIBURG

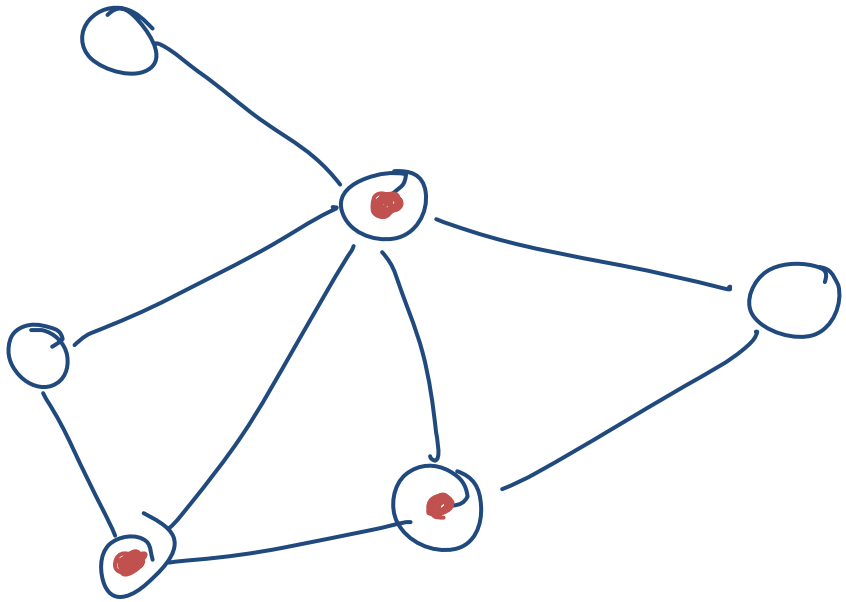
- Graphfärbungsproblem (graph coloring)
  - Färbe die Knoten eines gegebenen Graphen so ein, dass keine zwei benachbarten Knoten (= durch eine Kante verbunden) dieselbe Farbe haben
  - Die minimale Anzahl benötigter Farben nennt man die chromatische Zahl des Graphen
  - Die Berechnung der chromatischen Zahl eines Graphen ist ein sogenanntes NP-vollständiges Problem



Graphenprobleme 2 / 5

UNI FREIBURG

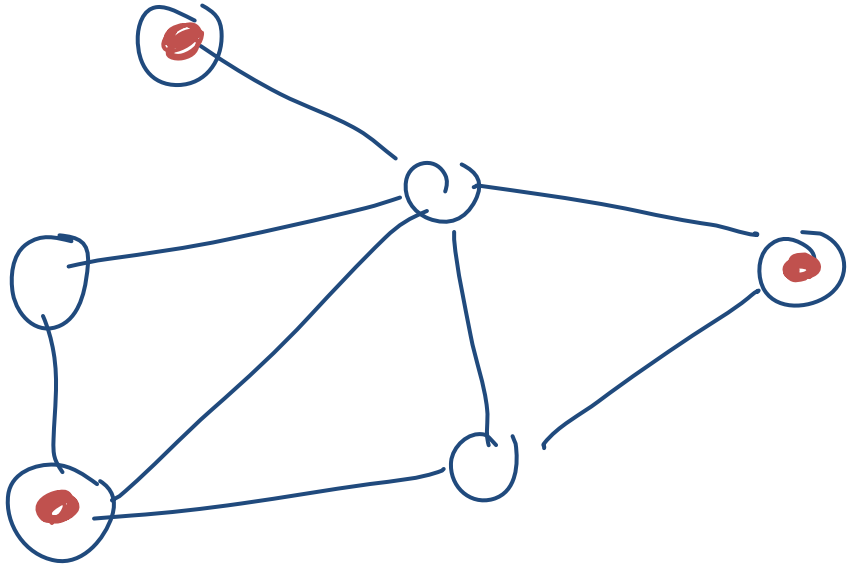
- Knotenüberdeckungen (vertex cover)
  - Für einen gegebenen Graphen G = (V, E) finde eine Teilmengen von Knoten V', so dass für jede Kante aus E mindestens einer ihrer beiden Endpunkte in V' liegt
  - Die Berechnung einer Knotenüberdeckung minimaler Größe ist ebenfalls ein NP-vollständiges Problem



Graphenprobleme 3 / 5

UNI FREIBURG

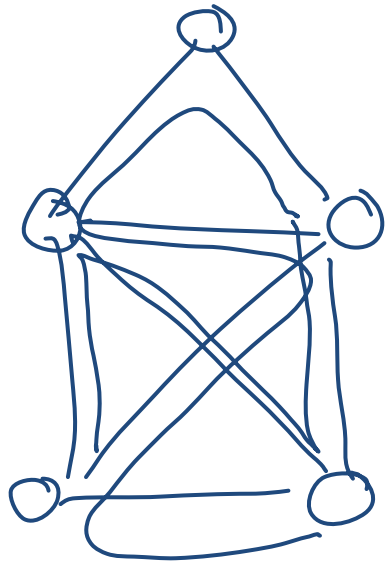
- Unabhängige Mengen (independent sets)
  - Für eine Graphen G = (V, E) finde eine Teilmenge von Knoten V', so dass zwischen keinem Paar u und v aus V' eine Kante aus E verläuft
  - Die Berechnung einer unabhängigen Menge maximaler Größe ist ebenfalls ein NP-vollständiges Problem



Graphenprobleme 4 / 5

UNI FREIBURG

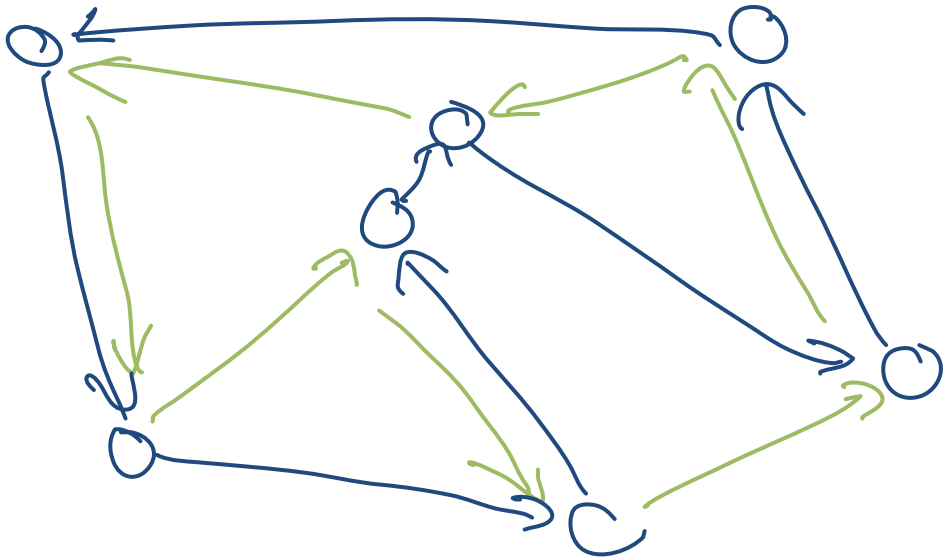
- Hamiltonscher Pfad / Eulerpfad
  - Für einen Graph $G = (V, E)$ ist ein Pfad auf dem jeder Knoten aus V genau einmal vorkommt ein **Hamiltonscher Pfad** und ein Pfad auf dem jede Kante aus E genau einmal vorkommt ein **Eulerpfad**
  - Wenn der Pfad wieder am Anfangsknoten ankommt, spricht man von einer **Tour**



Graphenprobleme 5 / 5

UNI FREIBURG

- Handlungsreisendenproblem (traveling saleman p)
  - Für einen Graphen G = (V, E) mit Kantenlängen, finde die Hamiltonsche Tour kürzester Länge
  - Ebenfalls ein NP-vollständiges Problem



Graphexploration

UNI FREIBURG

- "Definition"
  - Gegeben ein Graph $G = (V, E)$ und ein Startknoten $s \in V$, besuche alle Knoten von V "der Reihe nach"
  - Breitensuche = in der Reihenfolge der "Entfernung" von s
    - englisch breadth first search = BFS
  - Tiefensuche = erstmal "möglichst weit weg" von s
    - englisch depth first search = DFS
  - Das ist kein "Problem" an sich, taucht aber oft als Teil / Subroutine von anderen Algorithmen auf
    - zum Beispiel in der Übungsaufgabe, zur Berechnung der Zusammenhangskomponenten

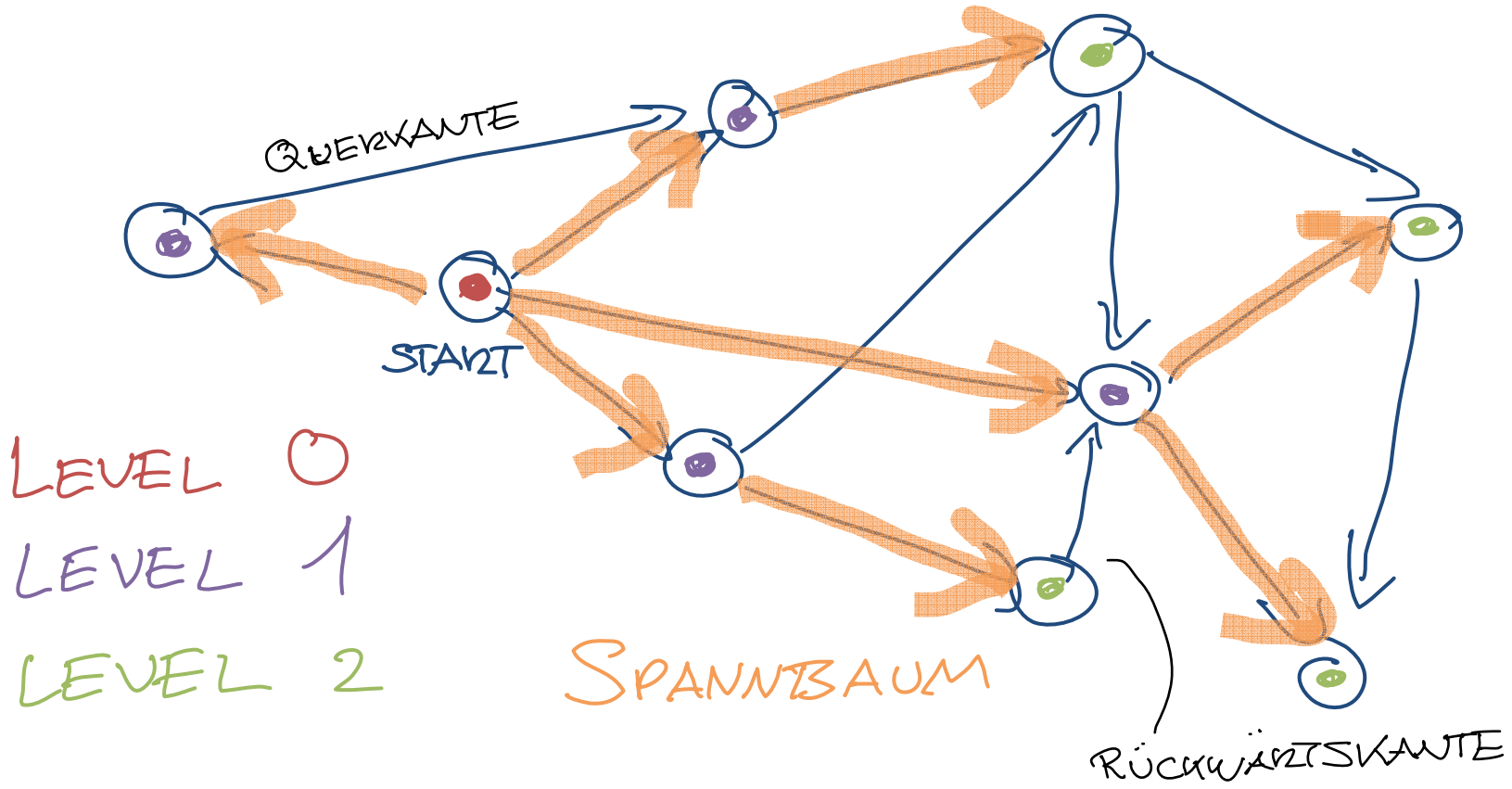
Breitensuche / Breadth First Search / BFS

UNI FREIBURG

- Idee
  - Markierung für jeden Knoten, zu Beginn alle unmarkiert
  - Beginne mit einem Startknoten und markiere ihn (Level 0)
  - Finde alle Knoten die zum Startknoten benachbart und noch nicht markiert sind und markiere sie (Level 1)
  - Finde alle Knoten, die zu einem Level-1 Knoten benachbart und noch nicht markiert sind und markiere sie (Level 2)
  - Usw. bis ein Level keine benachbarten Knoten mehr hat, die noch nicht markiert sind
  - Das markiert insbesondere alle Knoten, die in derselben Zusammenhangskomponente sind wie der Startknoten

Breitensuche / Breadth First Search / BFS

- Beispiel



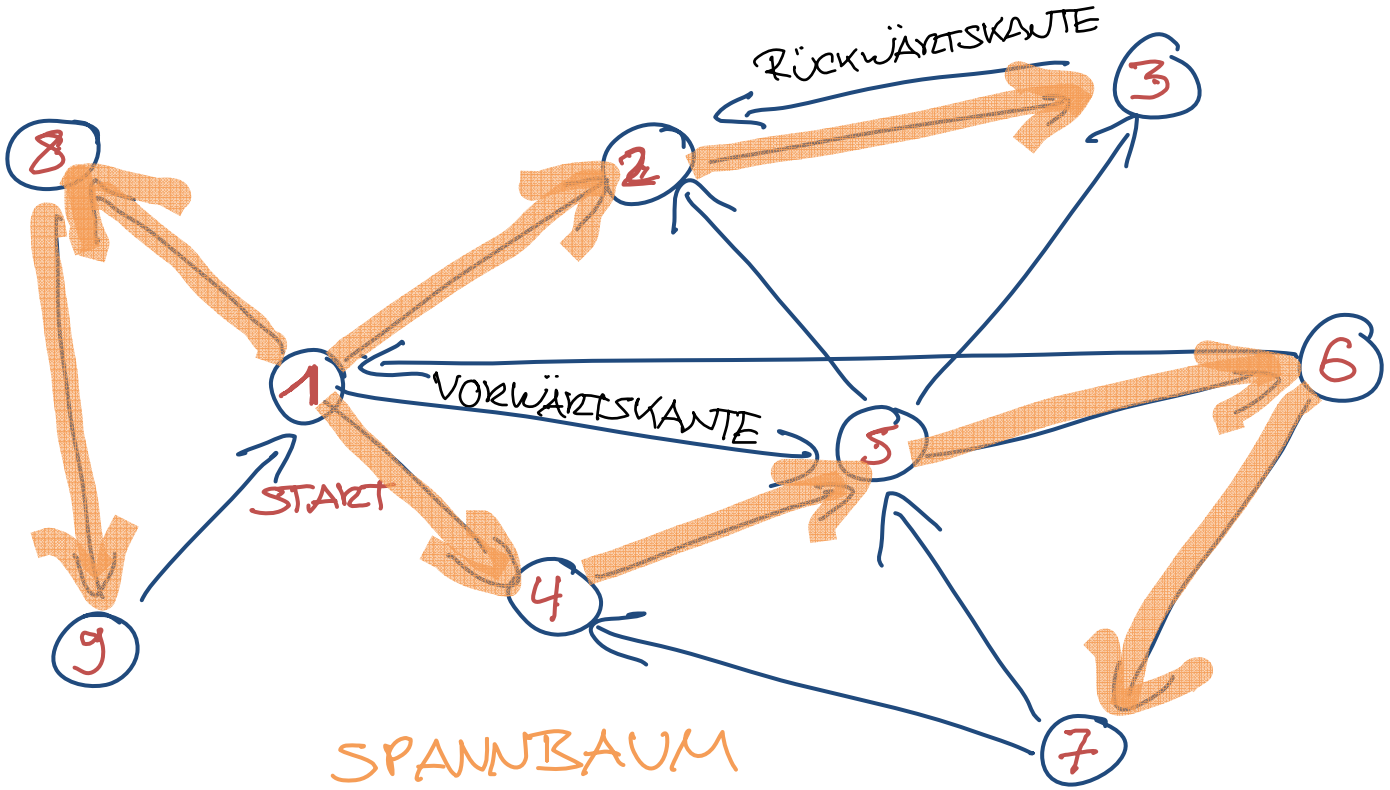
Tiefensuche / Depth First Search / DFS

UNI FREIBURG

- Idee
  - Markierung für jeden Knoten, zu Beginn alle unmarkiert
  - Beginne mit einem Startknoten und markiere ihn
  - Gehe in irgendeiner Reihenfolge die zum Startknoten benachbarten Knoten durch und tue folgendes
    - Falls der Knoten noch nicht markiert ist, markiere ihn und starte **rekursiv** eine Tiefensuche von dort aus
  - Das sucht zuerst "in die Tiefe" (vom Startknoten aus)
  - Auch Tiefensuche hat am Ende alle Knoten, die in derselben Zusammenhangskomponenten wie der Startknoten sind, markiert
  - Insbesondere kann man mit DFS **topologisch sortieren**

Tiefensuche / Depth First Search / DFS

- Beispiel



Komplexität von BFS und DFS

UNI FREIBURG

- Beide Verfahren ...
  - ... schauen sich jeden Knoten und jede Kante genau einmal an
  - Die Laufzeit ist also $O(|V| + |E|)$
  - Das kann man also (asymptotisch) nicht besser machen

Literatur / Links

- Graphen
  - In Mehlhorn/Sanders:

    8 Graph Representation
  - In Wikipedia

    http://en.wikipedia.org/wiki/Graph_(data_structure)
- Graphexploration
  - In Mehlhorn/Sanders:

    9 Graph Traversal
  - In Wikipedia

    http://en.wikipedia.org/wiki/Breadth-first_search

    http://en.wikipedia.org/wiki/Depth-first_search