

Jugendforum Informatik - Einführung

Tobias Heuer, Paul Jungeblut | 6. Februar 2020

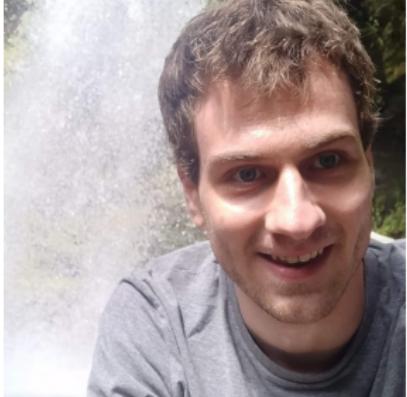
INSTITUT FÜR THEORETISCHE INFORMATIK



Wir sind...



Tobias Heuer



Paul Jungeblut

- beide ehemalige BwInf-Teilnehmer
- kennengelernt 2009 in Potsdam (BwInf-Workshop wie hier)
- seit 2012/2013 Informatikstudium in Karlsruhe (KIT)
- aktuell Doktoranden, Institut für theoretische Informatik

Die älteste Informatik-Fakultät Deutschlands:



- immer zum Wintersemester
- 6 Semester Bachelor
- 4 Semester Master

- Praktische Informatik
- Theoretische Informatik
- Technische Informatik
- Mathematik
- Nebenfach
(mehr Mathe, Physik, BWL, ...)



International Collegiate Programming Contest

- Programmierwettbewerb
- 3er-Teams
- 5 Stunden
- nur 1 Computer



Vorstellung
○○○●○

Graphen
○○○○○○

Aufgabe
○○○○○

Microsoft Imagine Cup

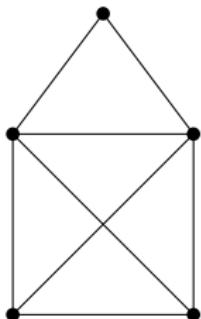
- Startup Wettbewerb
- Softwareentwicklung + Businessplan
- Über 10000 Teilnehmer weltweit jedes Jahr
- National Finals 2014 (München)
- World Finals 2016 (Seattle)



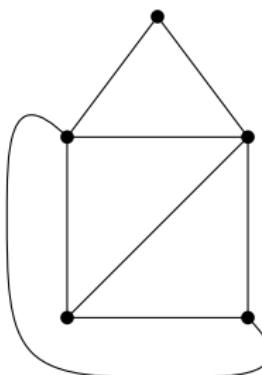
Definition

Ein **Graph** besteht aus

- einer Menge von **Knoten** (Objekten) und
- einer Menge von **Kanten** (Verbindungen zwischen zwei Objekten).



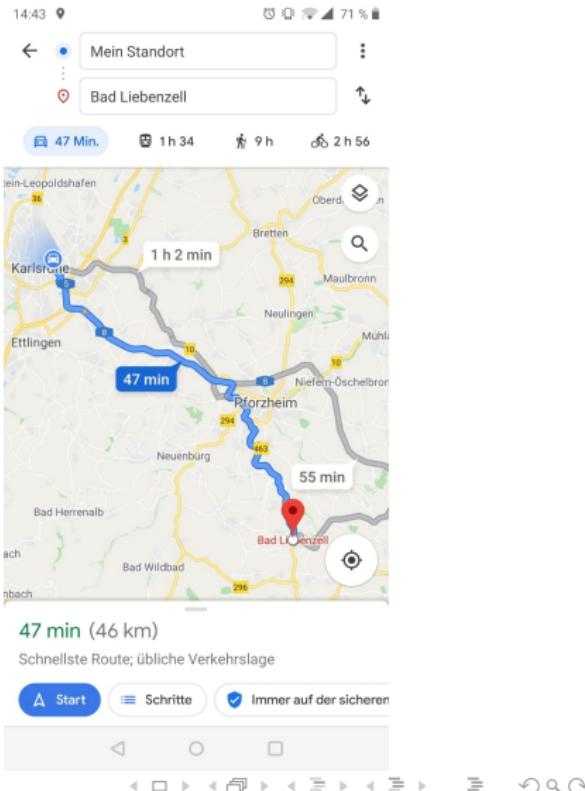
der gleiche
Graph auf
beiden Seiten



Graphen in der Praxis

Graphen werden benutzt für

- Routenplanung
- Paketvermittlung im Internet
- Informationsvisualisierung
- Spiele (Scotland Yard)
- und vieles, vieles mehr



Graphen: Notation

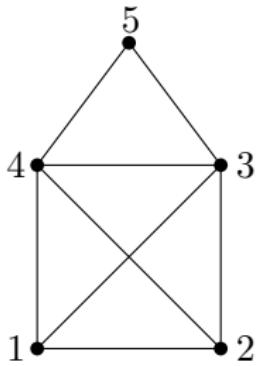
Graph $G = (V, E)$

- V : Knotenmenge (engl. Knoten = vertex)
- E : Kantenmenge (engl. Kante = edge)
- $N(v) = \{w \mid \{v, w\} \in E\}$: Nachbarschaft eines Knoten v .
- $d(v) = |N(v)|$: Grad eines Knoten v (Anzahl der Nachbarn)

Dazu noch ganz viele verwandte Konzepte:

- gewichteter Graph: Kanten haben eine Länge
- gerichteter Graph: Kanten sind asymmetrische Beziehungen
- Multigraph: mehrere Kanten zwischen zwei Knoten
- Hypergraph: eine Kante kann mehr als zwei Knoten umfassen
- ...

Graphen im Computer



Adjazenzmatrix: (2D-Array)

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

Kantenliste:

$\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\},$
 $\{2, 4\}, \{3, 4\}, \{3, 5\}, \{4, 5\}$

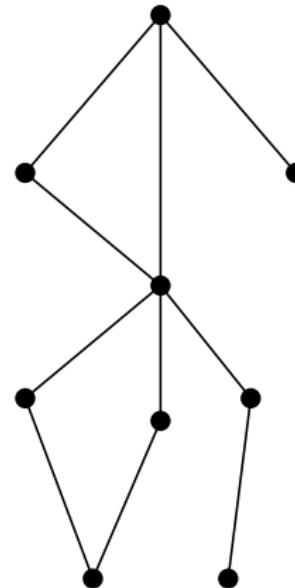
Adjazenzliste:

1 : 2, 3, 4
2 : 1, 3, 4
3 : 1, 2, 4, 5
4 : 1, 2, 3, 5
5 : 3, 4

Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

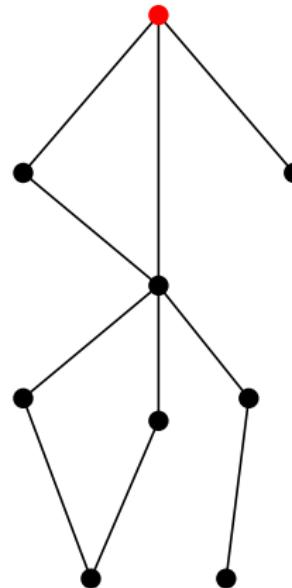
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

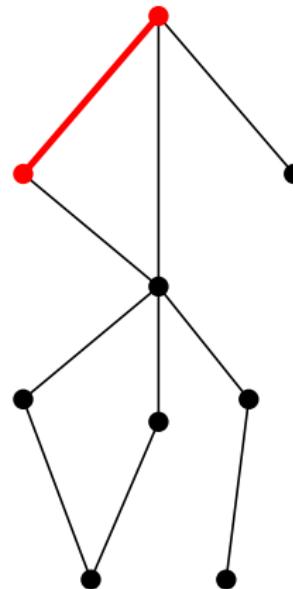
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

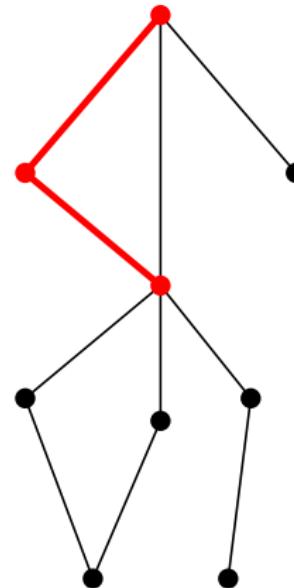
- ① Start bei beliebigem Knoten v
 - ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
 - ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

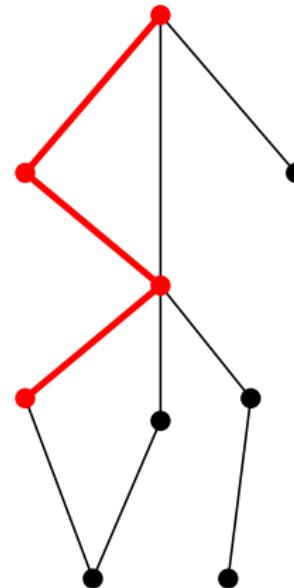
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

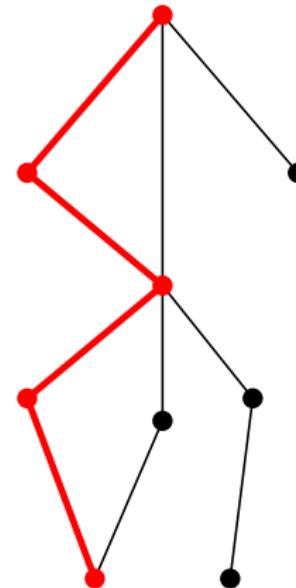
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

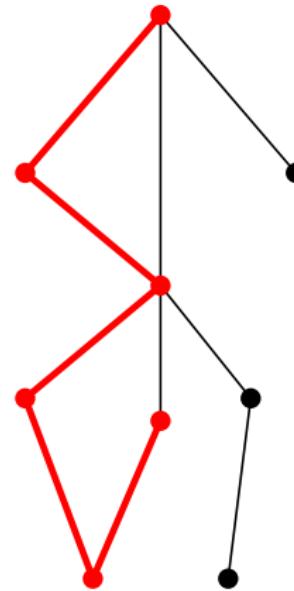
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

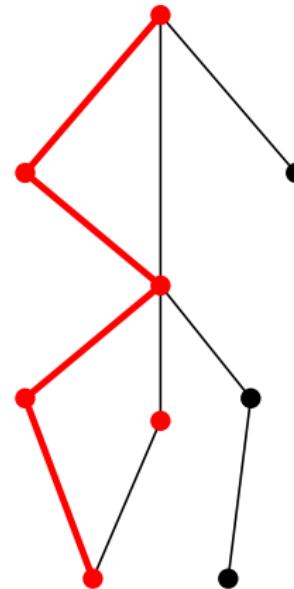
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

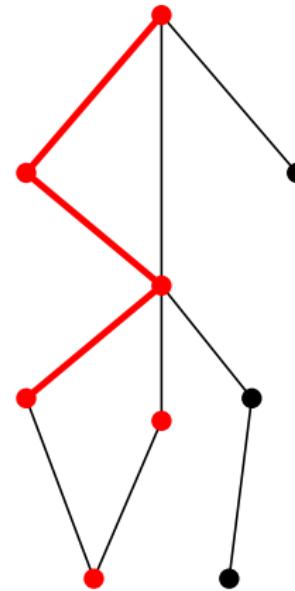
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

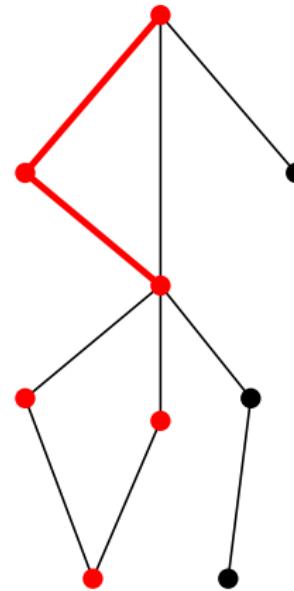
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

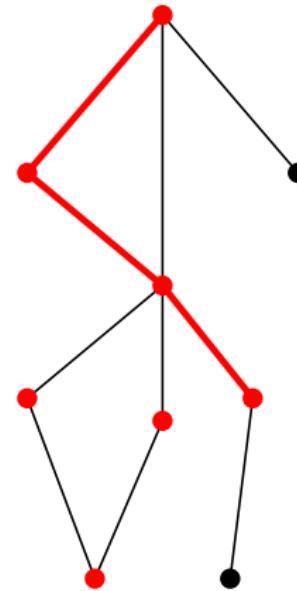
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

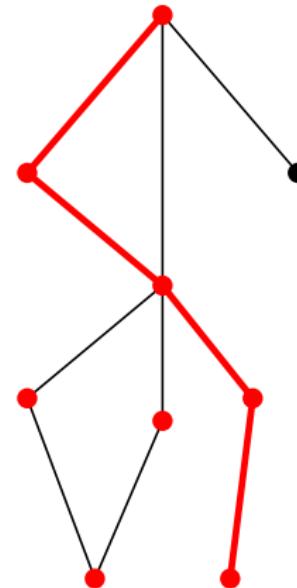
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

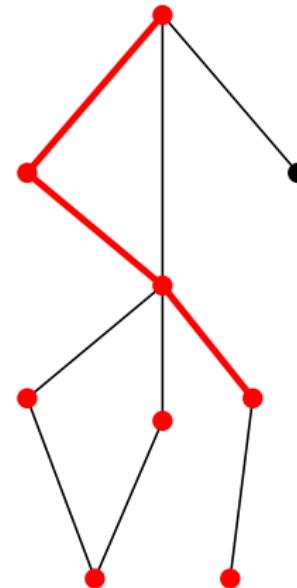
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

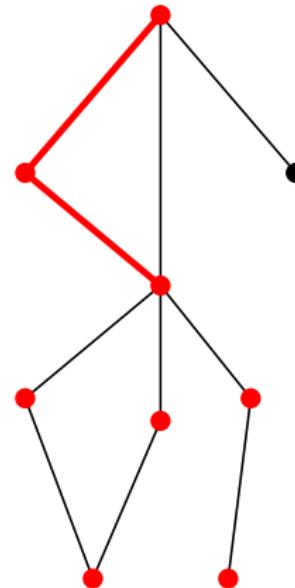
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

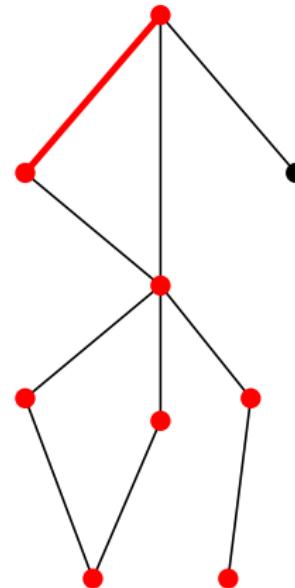
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

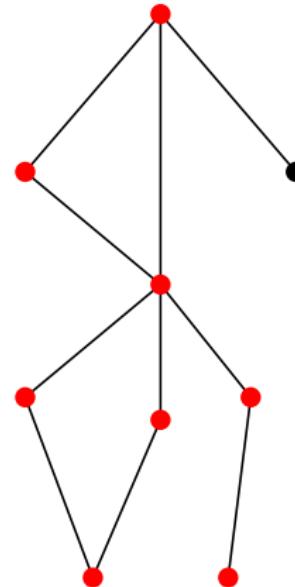
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

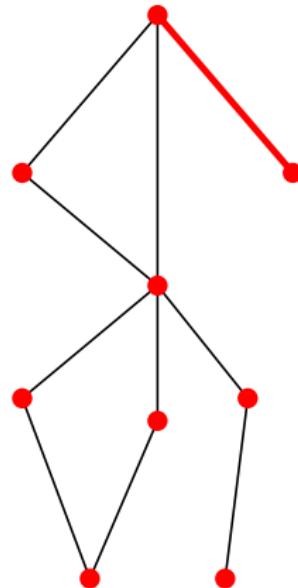
- ① Start bei beliebigem Knoten v
- ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
- ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche

Einfachste Art, alle Knoten zu besuchen:

- ① Start bei beliebigem Knoten v
 - ② v hat unbesuchten Nachbarn w :
 - besuche w
 - wiederhole Prozedur von w aus
 - ③ v hat keinen unbesuchten Nachbarn
 - gehe zurück zum Vorgänger



Tiefensuche: Code

```
void visit(int v) {  
    visited[v] = true;  
    for (each neighbor w of v) {  
        if (!visited[w]) visit(w);  
    }  
}
```

```
bool visited[n] = {false};  
for (int i = 0; i < n; ++i) {  
    if (!visited[i]) visit(i);  
}
```

Aufgabe

- Präsident Trumpf hat Angst vor *Fake News*
- NSA soll ihm helfen, alle Telefon-/Internetkabel abzuhören
- Dafür: mindestens einen Anschluss jeder Leitung überwachen

- Ziel: möglichst billige Lösung (wenige Anschlüsse anzapfen)

Aufgabe: Format

- 2-4er Teams
- mehrere Beispielnetzweke von uns
- versucht möglichst gute (optimale?) Lösung zu finden
Ansätze überlegen und programmieren
- heute Abend: jedes Team stellt seinen Ansatz und seine Ergebnisse vor

Fragen?

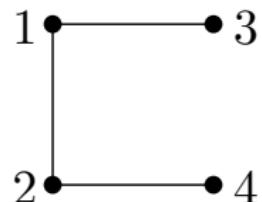
Wir sind den ganzen Tag da und helfen euch.
Sprecht uns einfach an.

Aufgabe: Eingabe

- ein Netzwerk pro Datei
`easy_1.graph`, `hard_3.graph`, ...
- verschiedene Größen

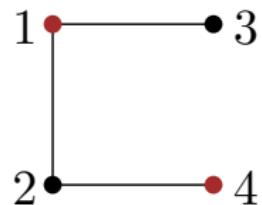
p td 4 3
1 2
1 3
2 4

- erste Zeile: p td n m
n: Anzahl Anschlüsse (1 ... n)
m: Anzahl Leitungen
- danache: eine Zeile pro Leitung: v w
zwischen Anschlüssen v und w



Aufgabe: Ausgabe

- eine Datei pro Lösung
 - easy_1.sol, hard_3.sol, ...
- erste Zeile: k
 - k: überwachte Anschlüsse
- danach: Liste mit Anschlüssen



Programm heute

jetzt: Einführung

bis 12:00 Arbeitsphase 1

12:00-13:00 Mittagessen

13:00-15:00 Arbeitsphase 2

15:00-15:30 Kaffeepause

15:30-17:00 Arbeitsphase 3

17:00-18:00 Präsentation der Ergebnisse

18:00 Abendessen