

Algorithmen und Datenstrukturen

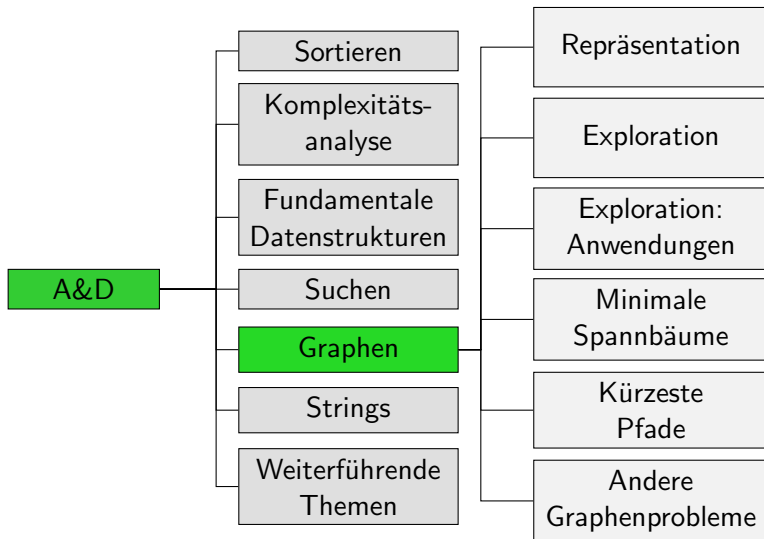
C1. Graphen: Grundlagen und Exploration

Gabriele Röger

Universität Basel

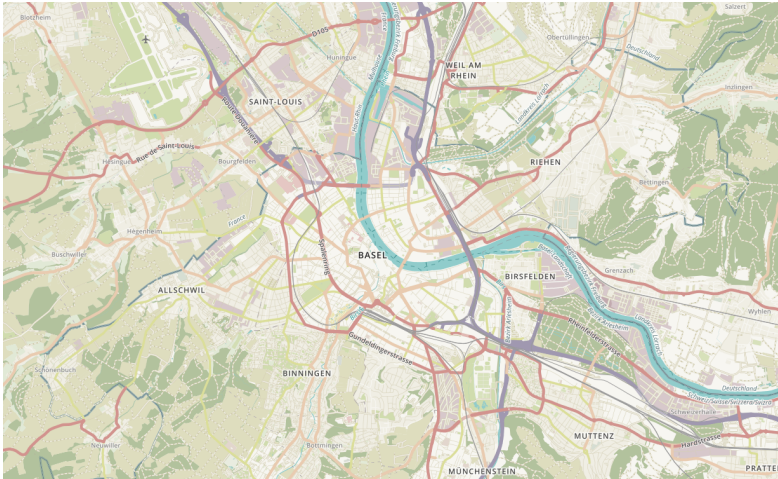
5. Mai 2022

Inhalt dieser Veranstaltung



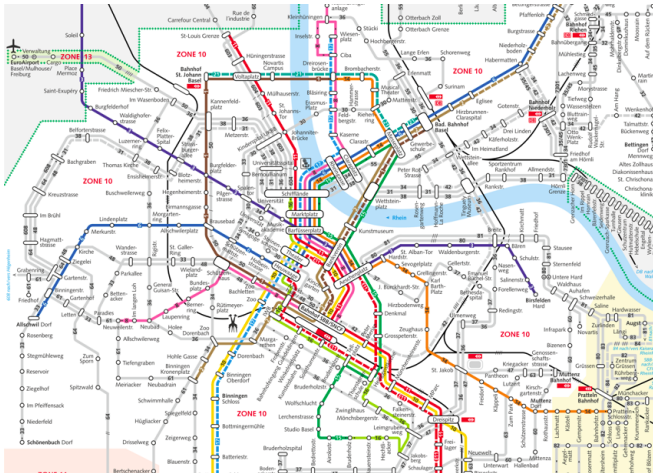
Motivation

Strassenkarten



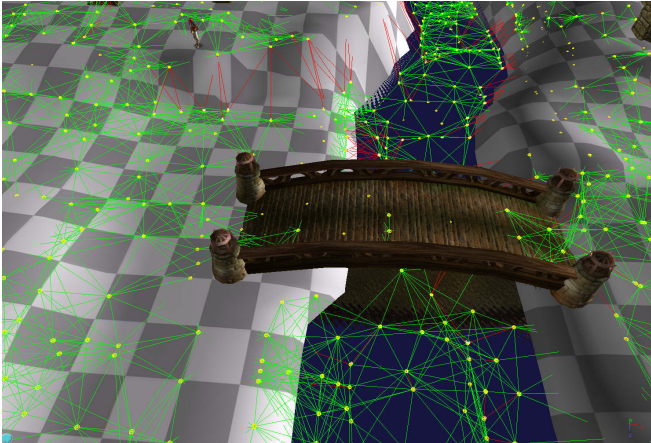
openstreetmap.org

Liniennetz



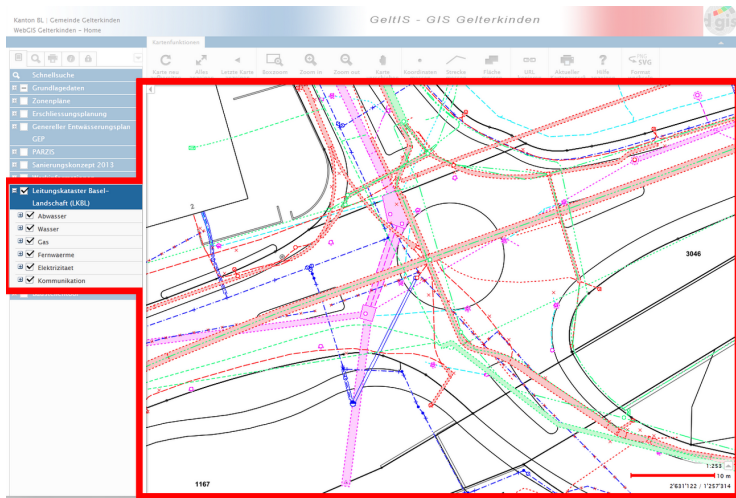
tnw.ch

Navigationsnetz in Spielen

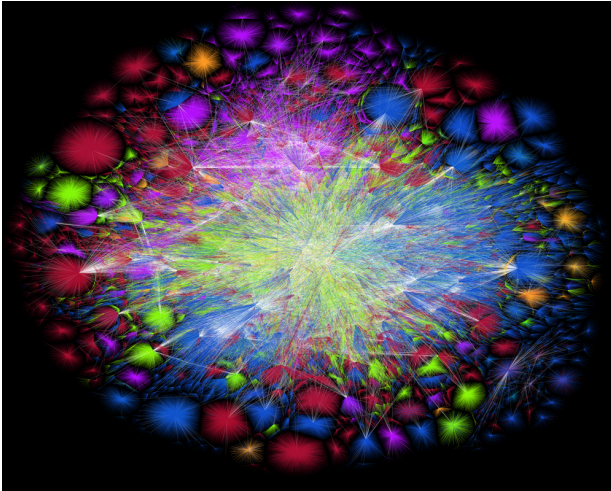


heroengine.com

Versorgungssystem



Internet



Barrett Lyon / The Opte Project
Visualization of the routing paths of the Internet.

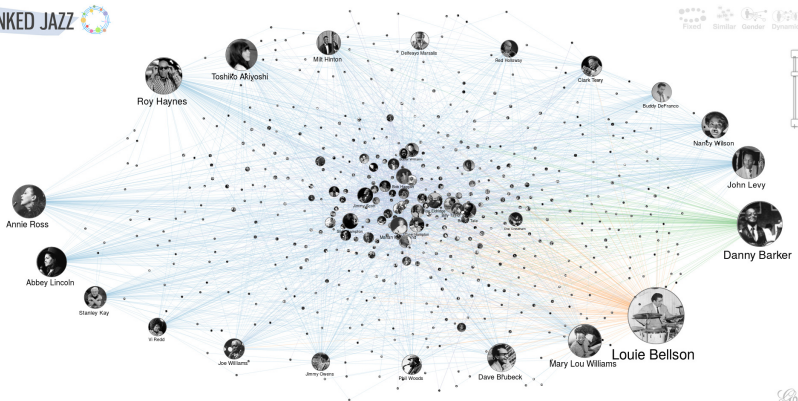
Soziale Netzwerke



„Visualizing Friendships“ von Paul Butler

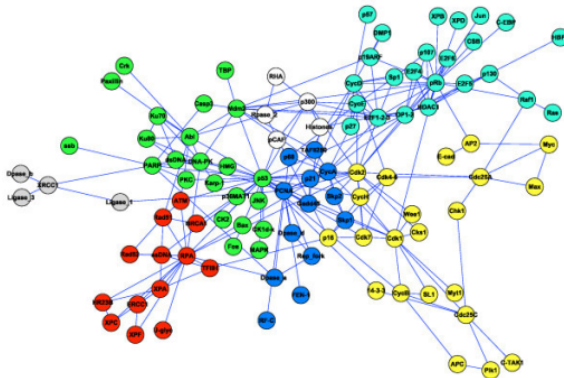
Zusammenarbeit

LINKED JAZZ 



linkedjazz.org

Protein-Interaktion



Network representation of the p53 protein interactions

Module detection in complex networks using integer optimisation,

Xu G, Bennett L, Papageorgiou LG, Tsoka S - Algorithms Mol Biol (2010)

Mögliche Fragestellungen

- Sind A und B verbunden?
- Was ist der kürzeste Weg zwischen A und B?
- Wie weit sind zwei Elemente höchstens voneinander entfernt?
- Wieviel Wasser kann die Kanalisation abführen?

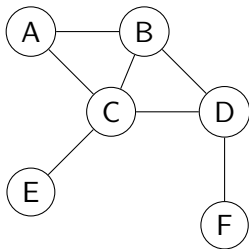
Abstrakte Graphen

Ein **Graph** besteht aus **Knoten** und **Kanten** zwischen Knoten.

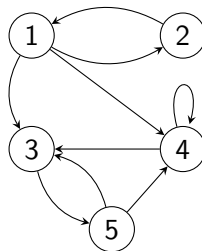
	Knoten	Kanten
Strassen	Kreuzung	Strassenabschnitt
Internet	AS ($\approx$ Provider)	Route
Facebook	Person	Freundschaft
Proteine	Protein	Interaktion

Grundlegende Definition

Ungerichtete und gerichtete Graphen



ungerichteter Graph



gerichteter Graph

Graphen

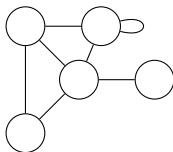
- Ein Graph besteht aus zwei Mengen **V** und **E**
 - **V**: Menge der **Knoten** (engl. vertices)
 - **E**: Menge der **Kanten** (engl. edges)
- Jede Kante verbindet zwei Knoten u und v
 - ungerichteter Graph: **Menge** $\{u, v\}$
 - gerichteter Graph: **Paar** (u, v)

Graphen

- Ein Graph besteht aus zwei Mengen **V** und **E**
 - **V**: Menge der **Knoten** (engl. vertices)
 - **E**: Menge der **Kanten** (engl. edges)
- Jede Kante verbindet zwei Knoten u und v
 - ungerichteter Graph: **Menge** $\{u, v\}$
 - gerichteter Graph: **Paar** (u, v)
- Bei **Multigraphen** kann es mehrere, parallele Kanten zwischen den gleichen Knoten geben.
- Bei **gewichteten** Graphen hat jede Kante ein Gewicht (Zahl).

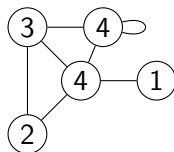
Ungerichtete Graphen: Terminologie

- **Nachbarn** eines Knotens u : alle Knoten v mit $\{u, v\} \in E$.



Ungerichtete Graphen: Terminologie

- **Nachbarn** eines Knotens u : alle Knoten v mit $\{u, v\} \in E$.
- $\text{degree}(v)$: **Grad** eines Knotens = **Anzahl der Nachbarn**.
 - Ausnahme: **Schleife** erhöht den Grad um 2.
Schleife = Kante, die einen Knoten mit sich selbst verbindet.



Gerichtete Graphen: Terminologie

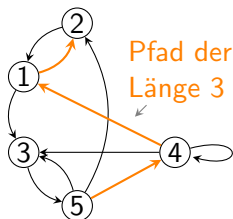
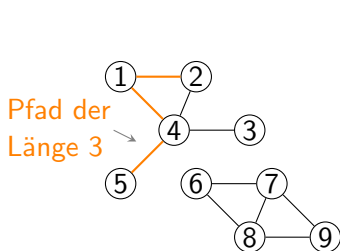
- **Nachfolger** eines Knotens u : alle Knoten v mit $(u, v) \in E$.
- **Vorgänger** eines Knotens u : alle Knoten v mit $(v, u) \in E$.

Gerichtete Graphen: Terminologie

- **Nachfolger** eines Knotens u : alle Knoten v mit $(u, v) \in E$.
- **Vorgänger** eines Knotens u : alle Knoten v mit $(v, u) \in E$.
- $\text{outdegree}(v)$: **Ausgangsgrad** = Anzahl der **Nachfolger**
- $\text{indegree}(v)$: **Eingangsgrad** = Anzahl der **Vorgänger**

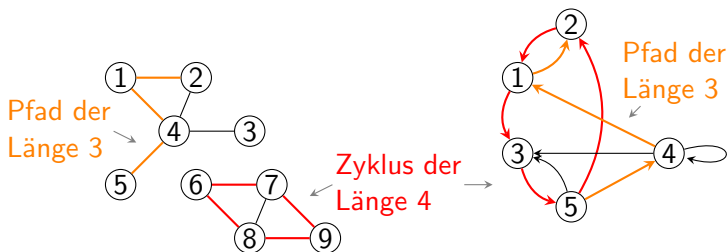
Pfade und Zyklen

- **Pfad der Länge n :** Sequenz $(v_0, \dots, v_n)$ von Knoten mit
 - $\{v_i, v_{i+1}\} \in E$ für $i = 0, \dots, n-1$ (ungerichteter Graph)
 - $(v_i, v_{i+1}) \in E$ für $i = 0, \dots, n-1$ (gerichteter Graph)
 - Beispiel: (5,4,1,2)



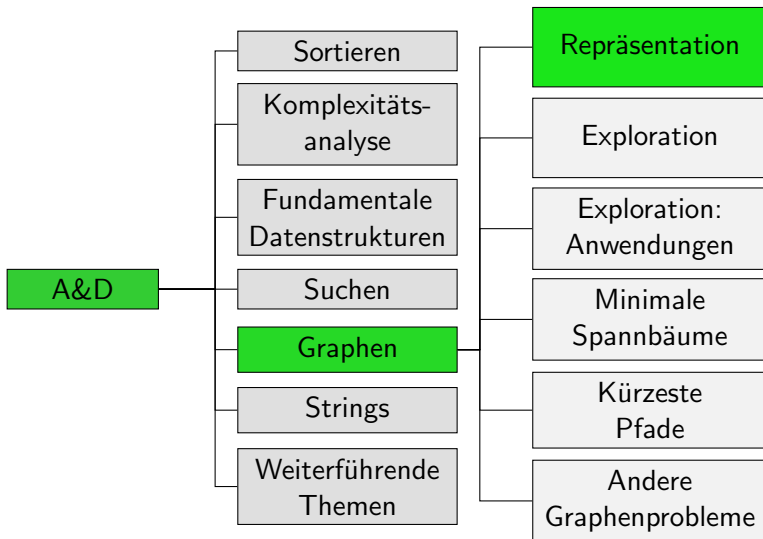
Pfade und Zyklen

- **Pfad der Länge n :** Sequenz $(v_0, \dots, v_n)$ von Knoten mit
 - $\{v_i, v_{i+1}\} \in E$ für $i = 0, \dots, n-1$ (ungerichteter Graph)
 - $(v_i, v_{i+1}) \in E$ für $i = 0, \dots, n-1$ (gerichteter Graph)
 - Beispiel: $(5, 4, 1, 2)$
- **Zyklus:** Pfad mit gleichem Start- und Endknoten, der keine Kante mehrmals verwendet.
 - $(6, 7, 9, 8, 6)$ im ungerichteten und $(5, 2, 1, 3, 5)$ im gerichteten Beispielgraphen
 - existiert kein Zyklus, ist der Graph **azyklisch**



Repräsentation

Inhalt dieser Veranstaltung



Repräsentation der Knoten

- Wir verwenden Zahlen von 0 bis $|V| - 1$ für die Knoten.
- Falls in Anwendung nicht gegeben: Verwende Symboltabellen, um zwischen Namen und Zahlen zu konvertieren.

Graphenrepräsentation mit Adjazenzmatrix

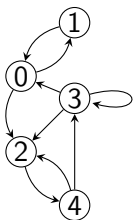
Graph $G = (\{0, \dots, |V| - 1\}, E)$ repräsentiert durch
 $|V| \times |V|$ -Matrix mit Einträgen a_{ik} (in Zeile i , Spalte k):

$$a_{ik} = \begin{cases} 1 & \text{falls } (i, k) \in E \text{ (gerichteter Graph) bzw.} \\ & \{i, k\} \in E \text{ (ungerichteter Graph)} \\ 0 & \text{sonst} \end{cases}$$

Graphenrepräsentation mit Adjazenzmatrix

Graph $G = (\{0, \dots, |V| - 1\}, E)$ repräsentiert durch
 $|V| \times |V|$ -Matrix mit Einträgen a_{ik} (in Zeile i , Spalte k):

$$a_{ik} = \begin{cases} 1 & \text{falls } (i, k) \in E \text{ (gerichteter Graph) bzw.} \\ & \{i, k\} \in E \text{ (ungerichteter Graph)} \\ 0 & \text{sonst} \end{cases}$$

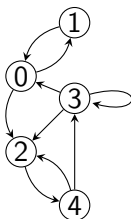


$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Graphenrepräsentation mit Adjazenzmatrix

Graph $G = (\{0, \dots, |V| - 1\}, E)$ repräsentiert durch
 $|V| \times |V|$ -Matrix mit Einträgen a_{ik} (in Zeile i , Spalte k):

$$a_{ik} = \begin{cases} 1 & \text{falls } (i, k) \in E \text{ (gerichteter Graph) bzw.} \\ & \{i, k\} \in E \text{ (ungerichteter Graph)} \\ 0 & \text{sonst} \end{cases}$$

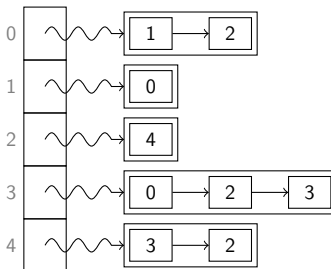
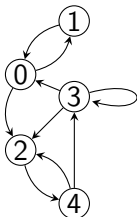


$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Bei ungerichteten
Graphen symmetrisch

Graphenrepräsentation mit Adjazenzliste

Speichere für jeden Knoten die Liste aller Nachfolger / Nachbarn



Repräsentation: Komplexität

	Adj.matrix	Adj.liste
Platzbedarf	$ V ^2$	$ E + V $
Kante hinzufügen	1	1
Kante zwischen u und v ?	1	$(\text{out})\text{degree}(v)$
Iterieren über ausgeh. Kanten	$ V $	$(\text{out})\text{degree}(v)$

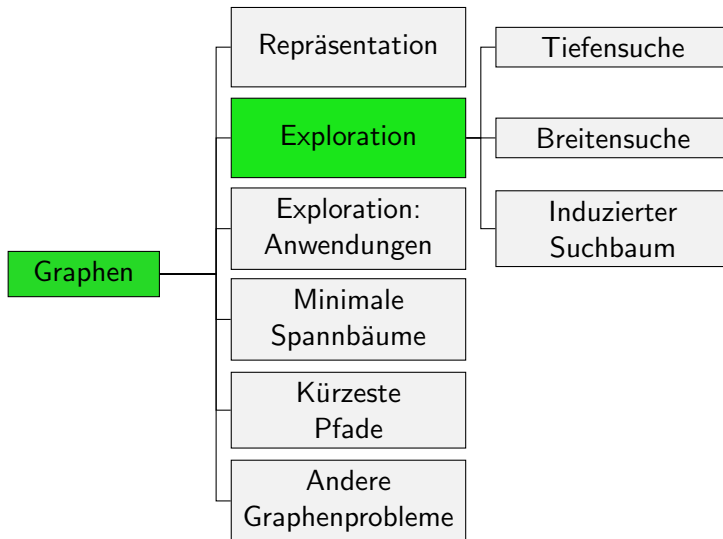
Repräsentation: Komplexität

	Adj.matrix	Adj.liste
Platzbedarf	$ V ^2$	$ E + V $
Kante hinzufügen	1	1
Kante zwischen u und v ?	1	(out)degree(v)
Iterieren über ausgeh. Kanten	$ V $	(out)degree(v)

Praxis: oft **dünne** Graphen (geringer durchschnittlicher Grad)
 Welche Repräsentation?

Graphenexploration

Graphen: Übersicht



Graphenexploration

- **Aufgabe:** Gegeben einen Knoten v , besuche alle Knoten, die von v aus erreichbar sind.
- Wird oft als Teil anderer Graphenalgorithmen benötigt.
- **Tiefensuche:** erst einmal möglichst tief in den Graphen (weit weg von v)
- **Breitensuche:** erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

Tiefensuche

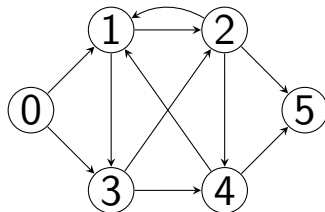
Markiere erreichte Knoten

- Markiere v
- Iteriere über die Nachfolger/Nachbarn w von v .
 - Falls w nicht markiert, starte rekursiv von w .

Englisch: **Depth-first search, DFS**

Tiefensuche: Beispiel

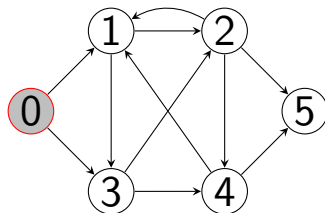
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Tiefensuche mit Startknoten 0
markiert Knoten in Reihenfolge

Tiefensuche: Beispiel

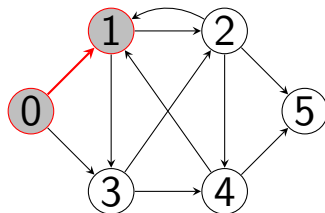
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Tiefensuche mit Startknoten 0
markiert Knoten in Reihenfolge
0

Tiefensuche: Beispiel

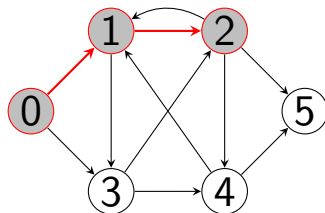
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Tiefensuche mit Startknoten 0
markiert Knoten in Reihenfolge
0 - 1

Tiefensuche: Beispiel

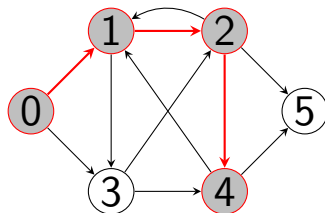
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Tiefensuche mit Startknoten 0
markiert Knoten in Reihenfolge
0 - 1 - 2

Tiefensuche: Beispiel

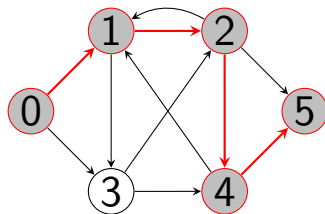
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Tiefensuche mit Startknoten 0
markiert Knoten in Reihenfolge
0 - 1 - 2 - 4

Tiefensuche: Beispiel

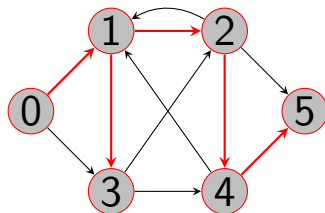
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Tiefensuche mit Startknoten 0
markiert Knoten in Reihenfolge
0 - 1 - 2 - 4 - 5

Tiefensuche: Beispiel

Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Tiefensuche mit Startknoten 0
markiert Knoten in Reihenfolge
0 - 1 - 2 - 4 - 5 - 3

Tiefensuche: Algorithmus (rekursiv)

```
1 def depth_first_exploration(graph, node, visited=None):
2     if visited is None:
3         visited = set()
4     if node in visited:
5         return
6     visited.add(node)
7     for s in graph.successors(node):
8         depth_first_exploration(graph, s, visited)
```

Tiefensuche: Algorithmus (rekursiv)

```
1 def depth_first_exploration(graph, node, visited=None):
2     if visited is None:
3         visited = set()
4     if node in visited:
5         return
6     visited.add(node)
7     for s in graph.successors(node):
8         depth_first_exploration(graph, s, visited)
```

Falls zu erwarten ist, dass ein Grossteil der Knoten besucht wird:
bool-Array statt Menge für visited

Depth-First-Knotenreihenfolgen

- **Preorder:** Knoten wird erfasst, bevor seine Kinder betrachtet werden.
- **Postorder:** Knoten wird erfasst, wenn die (rekursive) Tiefensuche mit allen seinen Kindern fertig ist.
- **Umgekehrte Postorder:** Wie Postorder, aber in umgekehrter Reihenfolge (spätere Knoten vorne)

```

1 def depth_first_exploration(graph, node):
2     if node in visited:
3         return
4     preorder.append(node)
5     visited.add(node)
6     for s in graph.successors(node):
7         depth_first_exploration(graph, s, visited)
8     postorder.append(node)
9     reverse_postorder.appendleft(node)

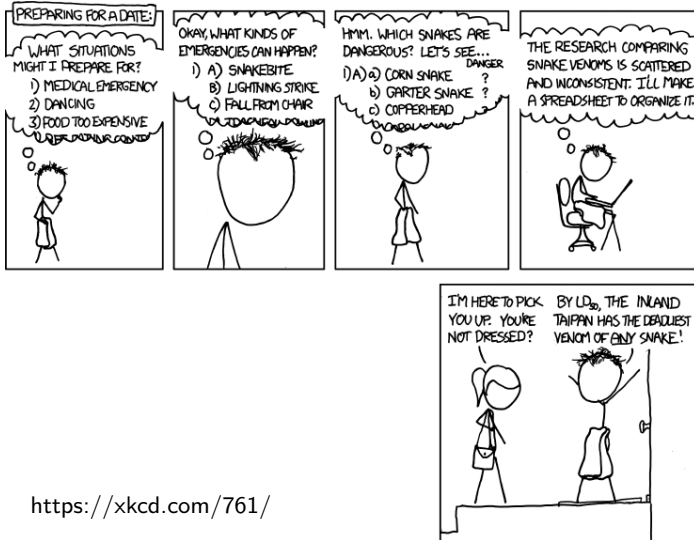
```

(Repräsentation der Knotenreihenfolgen als Deque)

Tiefensuche: Algorithmus (iterativ)

```
1 def depth_first_exploration(graph, node):
2     visited = set()
3     stack = deque()
4     stack.append(node)
5     while stack:
6         v = stack.pop() # LIFO
7         if v not in visited:
8             visited.add(v)
9             for s in graph.successors(v):
10                 stack.append(s)
```

Tiefensuche in der Praxis



<https://xkcd.com/761/>

I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

Breitensuche

Erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

Breitensuche

Erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

- Markiere v

→ Abstand 0

Englisch: Breadth-first search, BFS

Breitensuche

Erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

- Markiere v
→ Abstand 0
- Markiere alle unmarkierten Nachfolger/Nachbarn von v
→ Abstand 1

Englisch: Breadth-first search, BFS

Breitensuche

Erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

- Markiere v
→ Abstand 0
- Markiere alle unmarkierten Nachfolger/Nachbarn von v
→ Abstand 1
- Markiere alle unmarkierten Nachfolger/Nachbarn von Abstand-1-Knoten

Englisch: Breadth-first search, BFS

Breitensuche

Erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

- Markiere v
→ Abstand 0
- Markiere alle unmarkierten Nachfolger/Nachbarn von v
→ Abstand 1
- Markiere alle unmarkierten Nachfolger/Nachbarn von Abstand-1-Knoten
- Markiere alle unmarkierten Nachfolger/Nachbarn von Abstand-2-Knoten

Englisch: Breadth-first search, BFS

Breitensuche

Erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

- Markiere v
→ Abstand 0
- Markiere alle unmarkierten Nachfolger/Nachbarn von v
→ Abstand 1
- Markiere alle unmarkierten Nachfolger/Nachbarn von Abstand-1-Knoten
- Markiere alle unmarkierten Nachfolger/Nachbarn von Abstand-2-Knoten
- ...

Englisch: Breadth-first search, BFS

Breitensuche

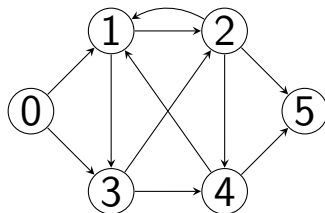
Erst alle Nachbarn, dann Nachbarn der Nachbarn, ...

- Markiere v
→ Abstand 0
- Markiere alle unmarkierten Nachfolger/Nachbarn von v
→ Abstand 1
- Markiere alle unmarkierten Nachfolger/Nachbarn von Abstand-1-Knoten
- Markiere alle unmarkierten Nachfolger/Nachbarn von Abstand-2-Knoten
- ...
- Bis Abstand- i -Knoten keine unmarkierten Nachfolger/Nachbarn haben

Englisch: Breadth-first search, BFS

Breitensuche: Beispiel

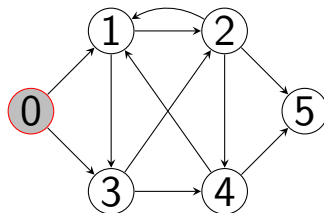
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Breitensuche mit Startknoten
0 markiert Knoten in
Reihenfolge

Breitensuche: Beispiel

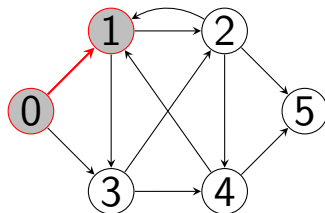
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Breitensuche mit Startknoten
0 markiert Knoten in
Reihenfolge
0

Breitensuche: Beispiel

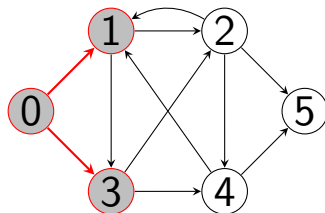
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Breitensuche mit Startknoten
0 markiert Knoten in
Reihenfolge
0 - 1

Breitensuche: Beispiel

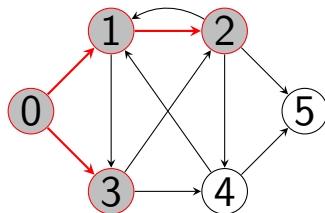
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Breitensuche mit Startknoten
0 markiert Knoten in
Reihenfolge
0 - 1 - 3

Breitensuche: Beispiel

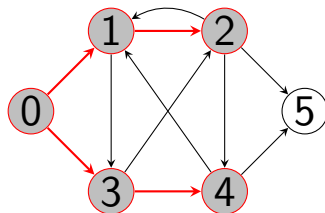
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Breitensuche mit Startknoten
0 markiert Knoten in
Reihenfolge
0 - 1 - 3 - 2

Breitensuche: Beispiel

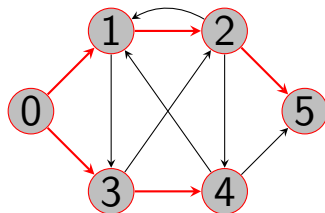
Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Breitensuche mit Startknoten
0 markiert Knoten in
Reihenfolge
0 - 1 - 3 - 2 - 4

Breitensuche: Beispiel

Hier: Besuche Nachfolger mit aufsteigender Knotennummer



Breitensuche mit Startknoten
0 markiert Knoten in
Reihenfolge
0 - 1 - 3 - 2 - 4 - 5

Breitensuche: Algorithmus (konzeptionell)

Einziger Unterschied zu iterativem Tiefensuchalgorithmus:

First-in-first-out-Behandlung der Knoten (statt last-in-first-out)

```
1 def breadth_first_exploration(graph, node):
2     visited = set()
3     queue = deque()
4     queue.append(node)
5     while queue:
6         v = queue.popleft() # FIFO
7         if v not in visited:
8             visited.add(v)
9             for s in graph.successors(v):
10                 queue.append(s)
```

Breitensuche: Algorithmus (etwas effizienter)

Nur erstes Antreffen eines Knotens wird weiterbetrachtet.

Wir können den Knoten direkt markieren und ihn bei einem weiteren Antreffen sofort verwerfen.

```
1 def breadth_first_exploration(graph, node):
2     visited = set()
3     queue = deque()
4     visited.add(node)
5     queue.append(node)
6     while queue:
7         v = queue.popleft()
8         for s in graph.successors(v):
9             if s not in visited:
10                 visited.add(s)
11                 queue.append(s)
```

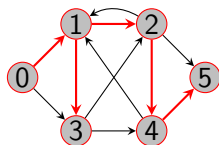
Laufzeit

Bei allen Algorithmenvarianten:

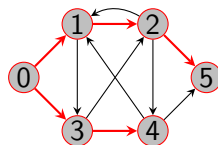
- Jeder erreichbare Knoten wird markiert.
- Man folgt jeder erreichbaren Kante einmal.
- Laufzeit $O(|V| + |E|)$
 - kann man auf erreichbare Knoten und Kanten einschränken

Induzierter Suchbaum

Der **induzierte Suchbaum** einer Graphenexploration enthält zu jedem besuchten Knoten (ausser dem Startknoten) eine Kante von dessen Vorgänger in der Exploration.



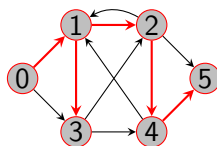
Tiefensuche



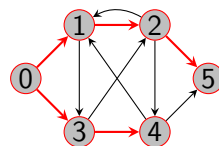
Breitensuche

Induzierter Suchbaum

Der **induzierte Suchbaum** einer Graphenexploration enthält zu jedem besuchten Knoten (ausser dem Startknoten) eine Kante von dessen Vorgänger in der Exploration.



Tiefensuche



Breitensuche

(induzierter Suchbaum $\neq$ binärer Suchbaum)

Induzierter Suchbaum: Beispiel Breitensuche

- Jeder Knoten hat höchstens einen Vorgänger im Baum.
- Repräsentiere induzierten Suchbaum durch Vorgängerrelation

Induzierter Suchbaum: Beispiel Breitensuche

- Jeder Knoten hat höchstens einen Vorgänger im Baum.
- Repräsentiere induzierten Suchbaum durch Vorgängerrelation
- Besuchte Knoten sind genau die, für die Vorgänger gesetzt ist:
Verzichte auf `visited`.

Induzierter Suchbaum: Beispiel Breitensuche

- Jeder Knoten hat höchstens einen Vorgänger im Baum.
- Repräsentiere induzierten Suchbaum durch Vorgängerrelation
- Besuchte Knoten sind genau die, für die Vorgänger gesetzt ist:
Verzichte auf `visited`.

```

1 def bfs_with_predecessors(graph, node):
2     predecessor = [None] * graph.no_nodes()
3     queue = deque()
4     # use self-loop for start node
5     predecessor[node] = node
6     queue.append(node)
7     while queue:
8         v = queue.popleft()
9         for s in graph.successors(v):
10             if predecessor[s] is None:
11                 predecessor[s] = v
12                 queue.append(s)

```

Zusammenfassung

- Graphen bestehen aus **Knoten** und **Kanten**
- Kanten können **gerichtet** oder **ungerichtet** sein.
- **Graphenexploration** besucht systematisch alle Knoten, die von einem bestimmten Knoten erreichbar sind.
 - **Tiefensuche** geht zuerst in die „Tiefe“.
 - **Breitensuche** besucht zuerst die Knoten, die näher am Startknoten sind.