

Programmierkurs Python II

Michaela Regneri
FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes

Sommersemester 2013



Übersicht

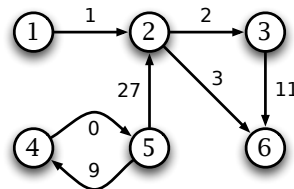
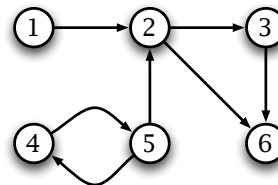
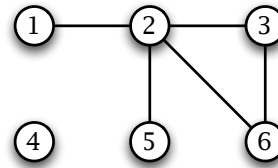
- Ein wenig Graph-Theorie (in aller Kürze)
- Datenstrukturen für Graphen
- Tiefen- und Breitensuche
- Nächste Woche: mehr Algorithmen

Was ist ein Graph?

Graphen kann man sich als Mengen von **Knoten** („Ecken“, nodes, vertices) vorstellen, die über **Kanten** (edges) verbunden sind

Varianten:

- *gerichtete* Graphen (Kanten haben eine Richtung)
- *gewichtete* Graphen (Kanten haben Gewichte)
- [...]



3

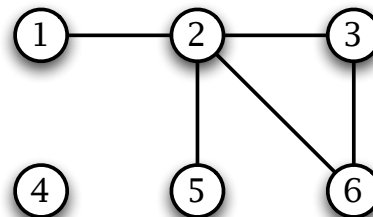
Ungerichteter (undirected) Graph

Ein **ungerichteter** Graph ist ein Paar (V, E) mit

- **V** eine Menge von Knoten
- **E** $\subseteq [V]^2$ eine Menge von Kanten (edges)

Zum Beispiel:

- $V = \{ 1, 2, 3, 4, 5, 6 \}$
- $E = \{ \{1, 2\}, \{2, 3\}, \{2, 5\}, \{2, 6\}, \{3, 6\} \}$



4

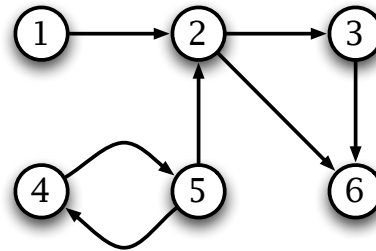
Gerichteter (directed) Graph

Ein **gerichteter** Graph ist ein Paar (V, E) mit

- V eine Menge von Knoten
- $E \subseteq V \times V$ eine Menge von Kanten

Zum Beispiel:

- $V = \{ 1, 2, 3, 4, 5, 6 \}$
- $E = \{ (1, 2), (2, 3), (5, 2), (2, 6), (3, 6), (4, 5), (5, 4) \}$



5

Inzident, adjazent

- Ein Knoten v ist mit einer Kante e **inzident** wenn v von e berührt wird (also Start- oder Endknoten von e ist).
- Zwei Knoten sind **adjazent** (benachbart), wenn sie über eine Kante verbunden sind.
- Der **Grad** eines Knotens = Anzahl inzidenter Kanten
- In gerichteten Graphen:
 - Eingangsgrad = Anzahl eingehender Kanten
 - Ausgangsgrad = Anzahl ausgehender Kanten

6

Pfade und Zyklen (paths and cycles)

- Ein **Pfad** ist eine Knotenfolge v_1, \dots, v_n , so dass gilt: es gibt Kante (v_i, v_{i+1}) , für alle $1 \leq i < n$
- Ein Pfad ist ein **Zyklus**, falls Start- und Endknoten identisch sind.
- Ein Pfad ist **einfach**, falls alle Knoten auf dem Pfad paarweise verschieden sind.
- Ein Graph heißt **zyklisch**, wenn er (mindestens) einen Zyklus enthält, ansonsten azyklisch.

7

Teilgraphen (subgraphs)

- Ist $G = (V, E)$ ein Graph, dann ist $G' = (V', E')$ ein **Teilgraph** von G falls $V' \subseteq V$ und $E' \subseteq E$.
- Der Teilgraph G' heißt **induziert** (aufgespannt), wenn er alle Kanten $\{v_1, v_2\} \in E$ mit $v_1, v_2 \in V'$ enthält.
 - Notation: $G' = G[V']$

8

Komponenten

- Ein Graph ist zusammenhängend (connected), wenn er für je zwei seiner Knoten einen Pfad enthält, der die Knoten verbindet.
- Ein maximal zusammenhängender Teilgraph eines Graphen G ist eine **Komponente** von G .
- Ein gerichteter Graph ist *schwach* zusammenhängend, wenn man einen zusammenhängenden Graphen erhält, indem man all seine Kanten durch ungerichtete Kanten ersetzt
- entsprechend: *schwache Zusammenhangskomponente* von gerichteten Graphen

9

Graphen als Datenstruktur

Graphen kann man auf verschiedene Weise als Datenstruktur repräsentieren:

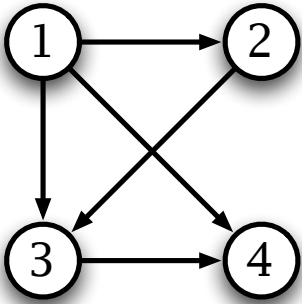
- **Adjazenzmatrix** (Nachbarschaftsmatrix)
- **Adjazenzliste** (Nachbarschaftsliste)
- [...]

10

Adjazenzmatrix

$|V| \times |V|$ Matrix A

- $A[i, j] = 1$ wenn $(i, j) \in E$
- $A[i, j] = 0$ wenn $(i, j) \notin E$

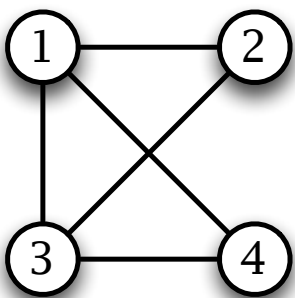


	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0

Adjazenzmatrix

$|V| \times |V|$ Matrix A

- $A[i, j] = 1$ wenn $\{i, j\} \in E$
- $A[i, j] = 0$ wenn $\{i, j\} \notin E$



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

Adjazenzmatrix

- $|V| \times |V|$ Matrix A
 - $A[i, j] = 1$ wenn $(i, j) \in E$
 - $A[i, j] = 0$ wenn $(i, j) \notin E$
- $O(|V|^2)$ Speicherplatz: gut geeignet für „dichte“ Graphen ($|E| \approx |V|^2$)
- $O(1)$ Zeit, um zu prüfen, ob $(i, j) \in E$

13

... in Python

```
class AdjacencyMatrix:
    def __init__(self, size):
        self.matrix = [ [ 0 ] * size for i in range(size) ]
    def add_edge(self, i, j):
        self.matrix[i][j] = 1
    def has_edge(self, i, j):
        return self.matrix[i][j]
    def edges(self):
        for (i, adj) in enumerate(self.matrix):
            for (j, b) in enumerate(adj):
                if b: yield (i, j)
    [...]
```

14

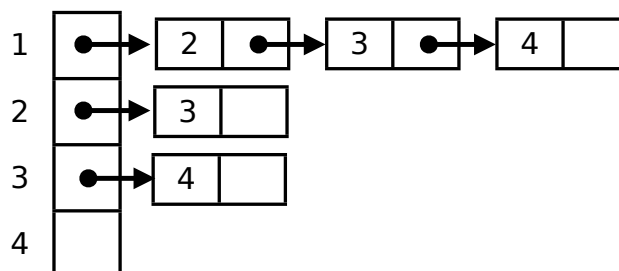
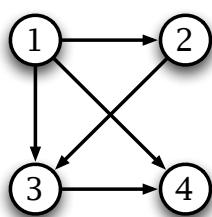
Alternativ ...

```
class AdjacencyMatrix:  
    def __init__(self, size):  
        self.matrix = [ 0 ] * (size * size)  
        self.size = size  
    def add_edge(self, i, j):  
        self.matrix[(i * self.size) + j] = 1  
    def has_edge(self, i, j):  
        return self.matrix[(i * self.size) + j]  
    def edges(self):  
        for i in range(self.size):  
            for j in range(self.size):  
                if self.has_edge(i, j): yield (i, j)  
[...]
```

15

Adjazenzlisten

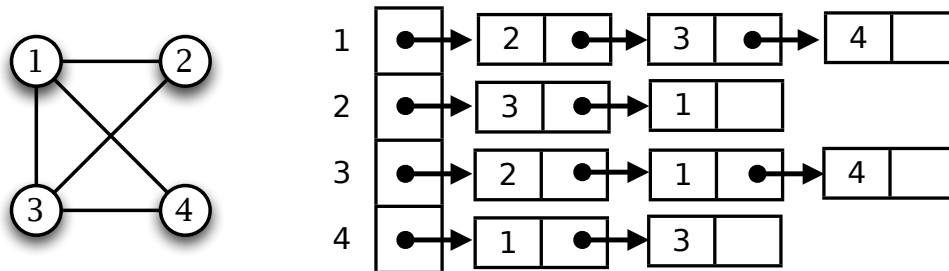
- Ein Array A von $|V|$ Listen
- $A[u]$ verweist auf alle adjazenten Knoten
- Python:
 - Array \approx Liste fester Länge
 - Liste \approx Liste variabler Länge (hier: verkettete Paare)



16

Adjazenzlisten

- Ein Array A von $|V|$ Listen
- $A[u]$ verweist auf alle adjazenten Knoten
- Python:
 - Array \approx Liste fester Länge
 - Liste \approx Liste variabler Länge (hier: verkettete Paare)



17

Adjazenzlisten

- Ein Array A von $|V|$ Listen
- $A[u]$ verweist auf alle adjazenten Knoten
- $O(|V| + |E|)$ Speicherplatz: gut geeignet für „dünne“ Graphen ($|E| \approx |V|$)
- $O(|V|)$ Zeit, um zu prüfen, ob $(i, j) \in E$

18

... in Python

```
class AdjacencyList:
    def __init__(self, size):
        self.node_list = [ [] for i in range(size) ]
    def add_edge(self, i, j):
        if not self.has_edge(i, j):
            self.node_list[i].append(j)
    def has_edge(self, i, j):
        return j in self.node_list[i]
    def edges(self):
        for (i, adj) in enumerate(self.node_list):
            for j in adj:
                yield (i, j)
    [...]
```

19

Graphen in Python

- Adjazenzlisten bzw. Matrizen sind nicht unbedingt die natürlichsten Datenstrukturen, um Graphen in Python zu implementieren.
- In Python bieten sich zwei Varianten an:
 - Graphen als Wörterbücher mit Mengen (oder Listen)
 - Graphen ganz objektorientiert (Knoten / Kanten in Klassen)
- Variante 1: ein Dictionary, das Knoten jeweils auf die Menge der adjazenten Knoten abbildet.
 - etwas langsamer, aber viel flexibler als die Matrizen
 - alle Datentypen, die als Schlüssel eines **dict** verwendet werden dürfen, können für Knoten stehen.

20

Zum Beispiel so

```
class Graph:
    def __init__(self):
        self.nodes = dict()

    def add_edge(self, i, j):
        self.nodes[i] = self.nodes.get(i, set()) | set([j])

    def has_edge(self, i, j):
        return j in self.nodes.get(i, [])

    def edges(self):
        for (i, adj) in self.nodes.items():
            for j in adj:
                yield (i, j)
```

21

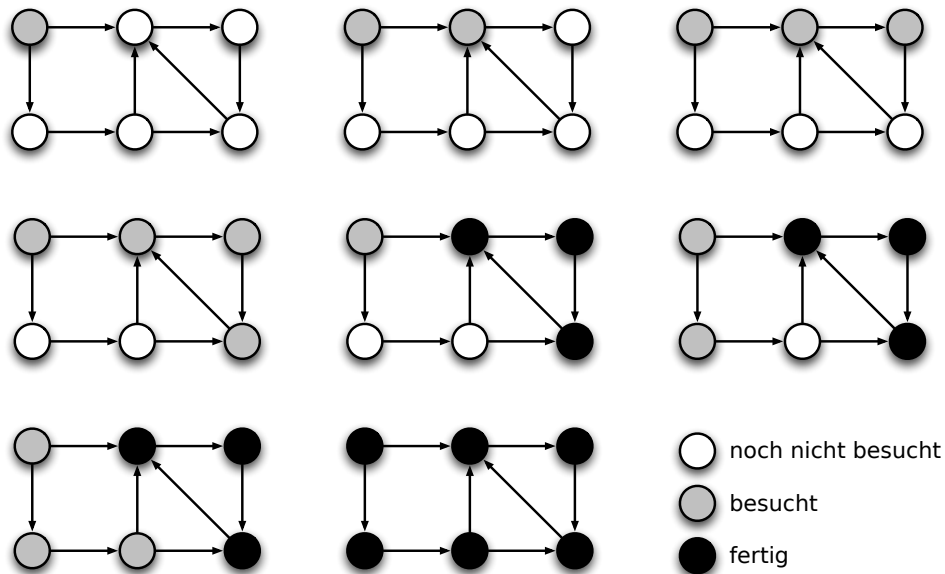
Suche in Graphen

Besuche alle Knoten eines Graphen:

- Breitensuche - zuerst alle Geschwister besuchen
- Tiefensuche - erst alle Kinder-Knoten, dann die Geschwister

22

Tiefensuche



23

Tiefensuche

```
def nop(node): pass
```

```
def dfs(self, node, dosomething = nop):  
    visited = set()  
    stack = [node]  
    while stack:  
        node = stack.pop()  
        if node not in visited:  
            visited.add(node)  
            dosomething(node)  
            stack.extend(self.children(node))
```

[...]

24

Tiefensuche (Iterator)

```
class Graph(object):
    def dfs(self, node):
        visited = set()
        stack = [node]
        while stack:
            node = stack.pop()
            if node not in visited:
                visited.add(node)
                yield node
                stack.extend(self.children(node))
        [...]
```

```
for n in g.dfs(start):
    dosomething(n)
```

25

Tiefensuche – Anwendungen

- Erreichbarkeit, transitiver Abschluss
- Untersuchung des Graphen auf Zyklen
- Zusammenhangskomponenten
- Topologische Sortierung
- [...]

26

Breitensuche

...wie Tiefensuche, nur mit einer Queue statt eines Stacks

- `queue = []`
- `queue.append(elt)` # am Ende anhängen
- `queue.pop(0)` # von Anfang entfernen + zurückgeben

Queues in python:

```
from collections import deque
```

```
my_queue = deque([])  
my_queue.append(...)  
my_queue.extend([...])  
my_queue.popleft()
```

langsamer!

```
my_queue = []  
my_queue.append(...)  
my_queue.extend([...])  
my_queue.pop(0)
```