

Programmierpraktikum

Sommersemester 2007 - 11.05.2007

Björn Wilmsmann, B.A.
Sprachwissenschaftliches Institut
Ruhr-Universität Bochum
wilmsmann@linguistics.rub.de

Heute: Graphen

- grundlegende Datenstruktur für semantisches Netze
- sowohl Strukturen im WordNet als auch im Kontext Semantic Web sind Graphen
- finden auch in vielen anderen Bereichen der Softwareentwicklung Verwendung (z.B. informationstechnische Implementation sozialer Netzwerke, Routenplaner)

Graphen

- Definition
- V (für vertices) ist eine endliche, nichtleere Menge von Knoten
- die Menge der Kanten E (für edges) ist definiert als $E \subseteq V \times V$

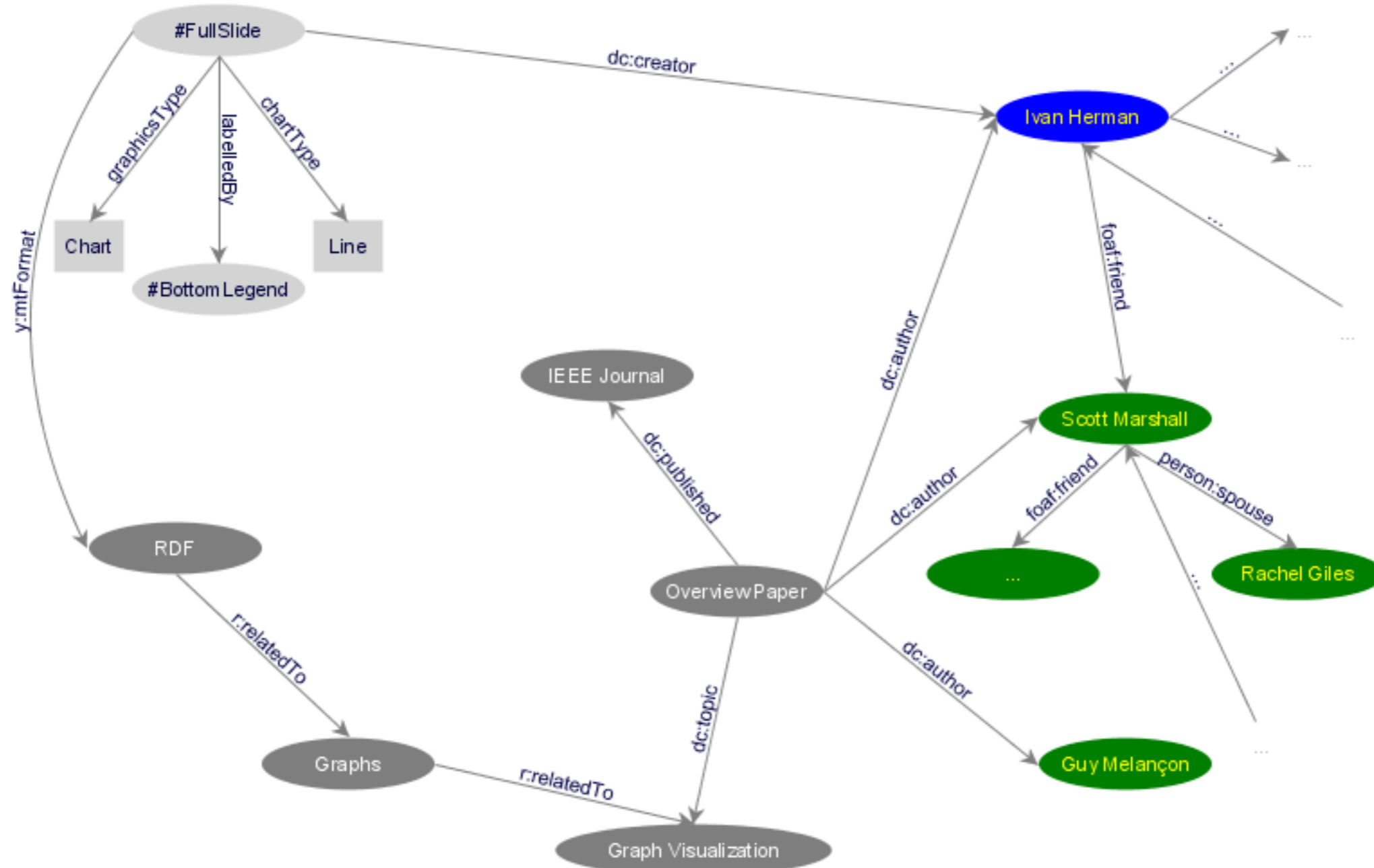
Graphen

- Definition
- bei ungerichteten Graphen ist die Relation E zusätzlich noch symmetrisch
- gewichtete Graphen ordnen jedem Element aus E zusätzlich eine Gewichtung zu

Graphen

- Beispiele
 - Person A kennt Person B (FOAF, Small World Hypothesis)
 - Es gibt einen Weg zwischen Ort A und Ort B
 - Wort A ist synonym zu Wort B

Beispiel



Graphen

- Speicherung
 - als Adjazenzmatrix
 - als Adjazenzliste

Intermission: O- Notation

- dient der Darstellung von Laufzeiten und Speicherbedarf für Datenstrukturen und Algorithmen
- Maß für den Grad des Laufzeitverhaltens / Speicherbedarf, nicht für das exakte Laufzeitverhalten / Speicherbedarf
- ‚eine genaue Art, etwas ungenaues auszudrücken‘

Intermission: O- Notation

- intuitive Aussage: Die zu untersuchende Funktion f wächst höchstens so schnell wie eine mit einem beliebigen, aber festen Faktor multiplizierte Funktion g
- falls der Grenzwert von $f(n) / g(n)$ für $n \rightarrow \infty$ existiert, dann ist $f = O(g)$

Laufzeitklassen

	Sprechweise	Beispiel
$O(1)$	konstant	Hash
$O(\log n)$	logarithmisch	Suchen in Menge
$O(n)$	linear	Editieren aller Elem.
$O(n \log n)$	-	Heapsort
$O(n^2)$	quadratisch	Dijkstra's Algorithm
$O(n^k)$	polynomiell	Floyd's Algorithm
$O(2^n)$	exponentiell	Backtracking

Adjazenzmatrix

- $n \times n$ Matrix A mit $A_{ij} = \text{true}$, falls $(i, j) \in E$ und $A_{ij} = \text{false}$, falls $(i, j) \notin E$
- Implementierung als mehrdimensionale Strukturen (in Perl: Hash-of-Hashes)
- bei gewichteten Graphen werden die Gewichtungen statt der booleschen Wert verwandt

Adjazenzmatrix

- Vorteil: Ob $(i, j) \in E$ lässt sich in $O(1)$ Zeit feststellen, Operationen aus der linearen Algebra anwendbar
- Nachteil: Platzbedarf von $O(n^2)$

Beispiel ungewichtet

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

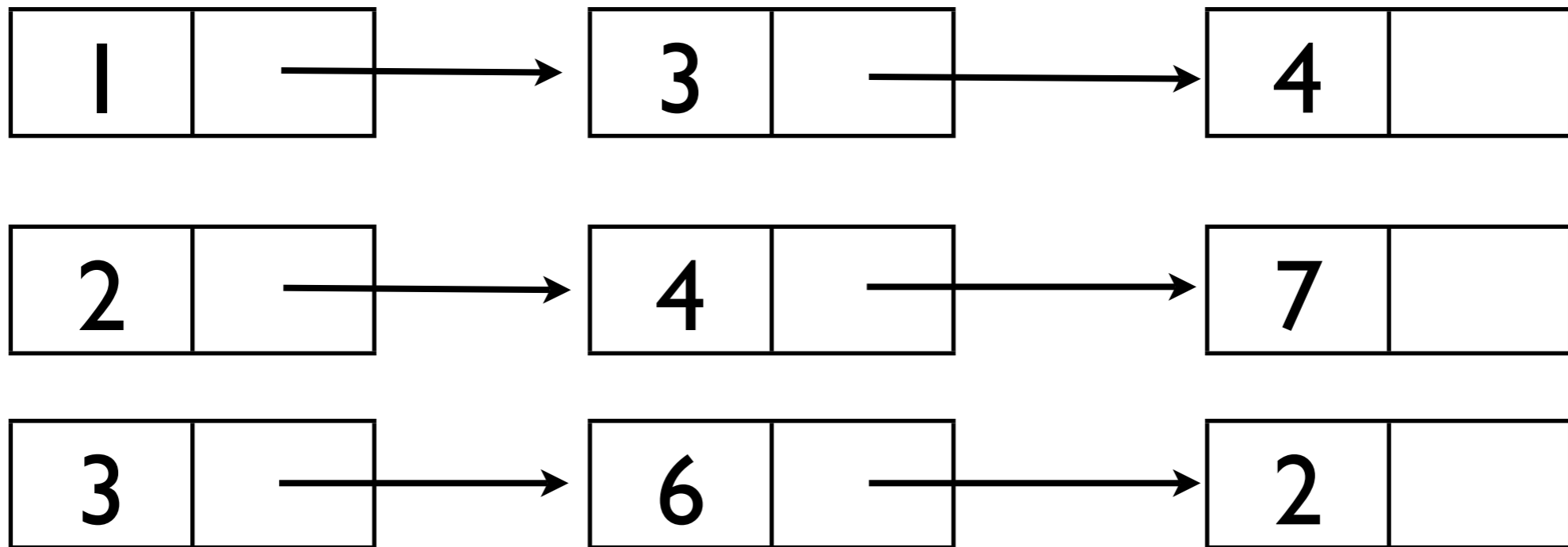
Beispiel gewichtet

	1	2	3	4	5
1	0	5	0	2	1
2	3	0	9	0	0
3	0	22	0	17	0
4	23	0	7	0	8
5	5	0	14	0	0

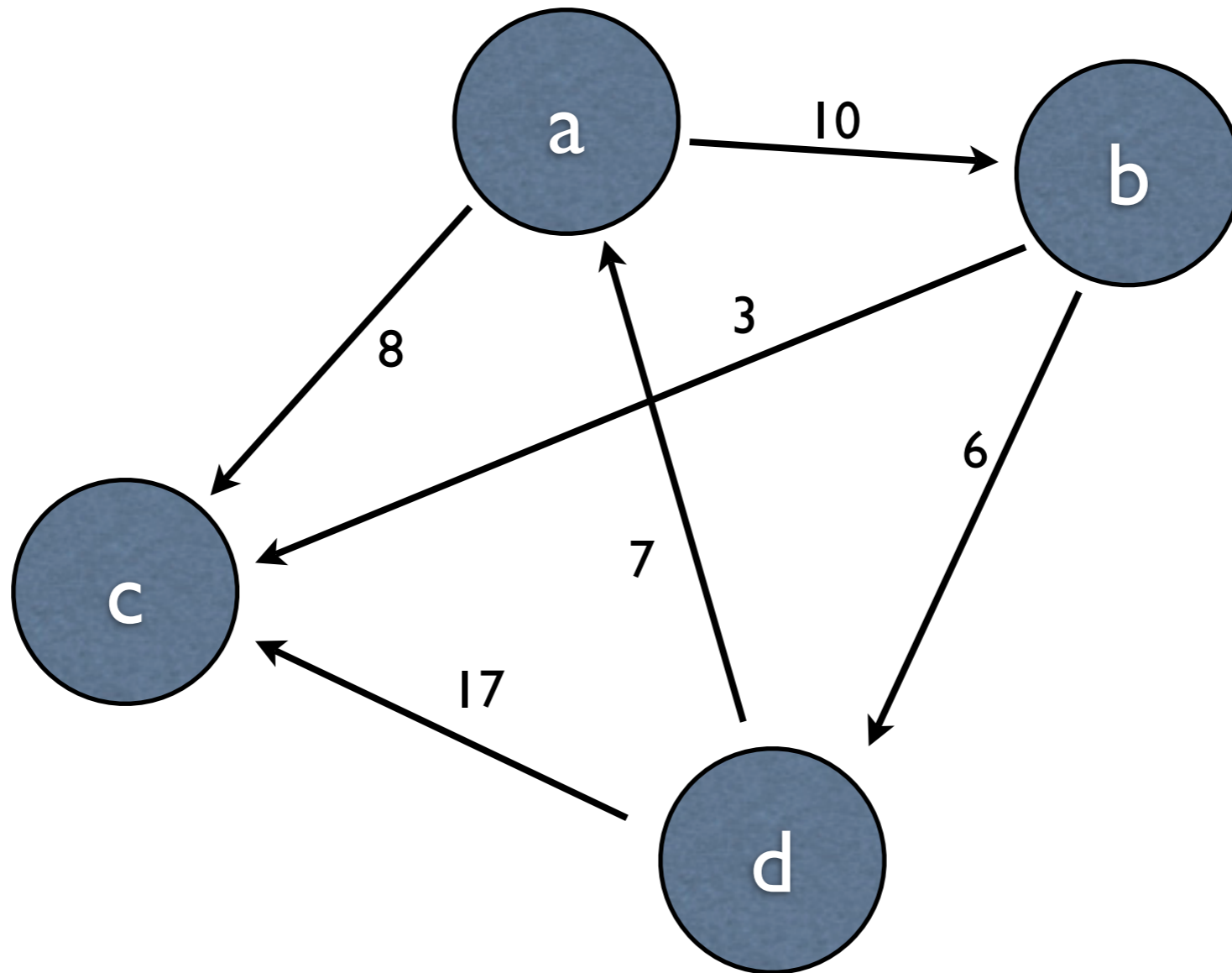
Adjazenzliste

- Verwaltung von Nachbarknoten per einfach verketteter Liste
- Vorteil: $O(n+e)$ Speicherbedarf mit $n = |V|$ und $e = |E|$
- Nachteil: $O(k)$ Zeitbedarf bei k Nachfolgerknoten

Beispiel



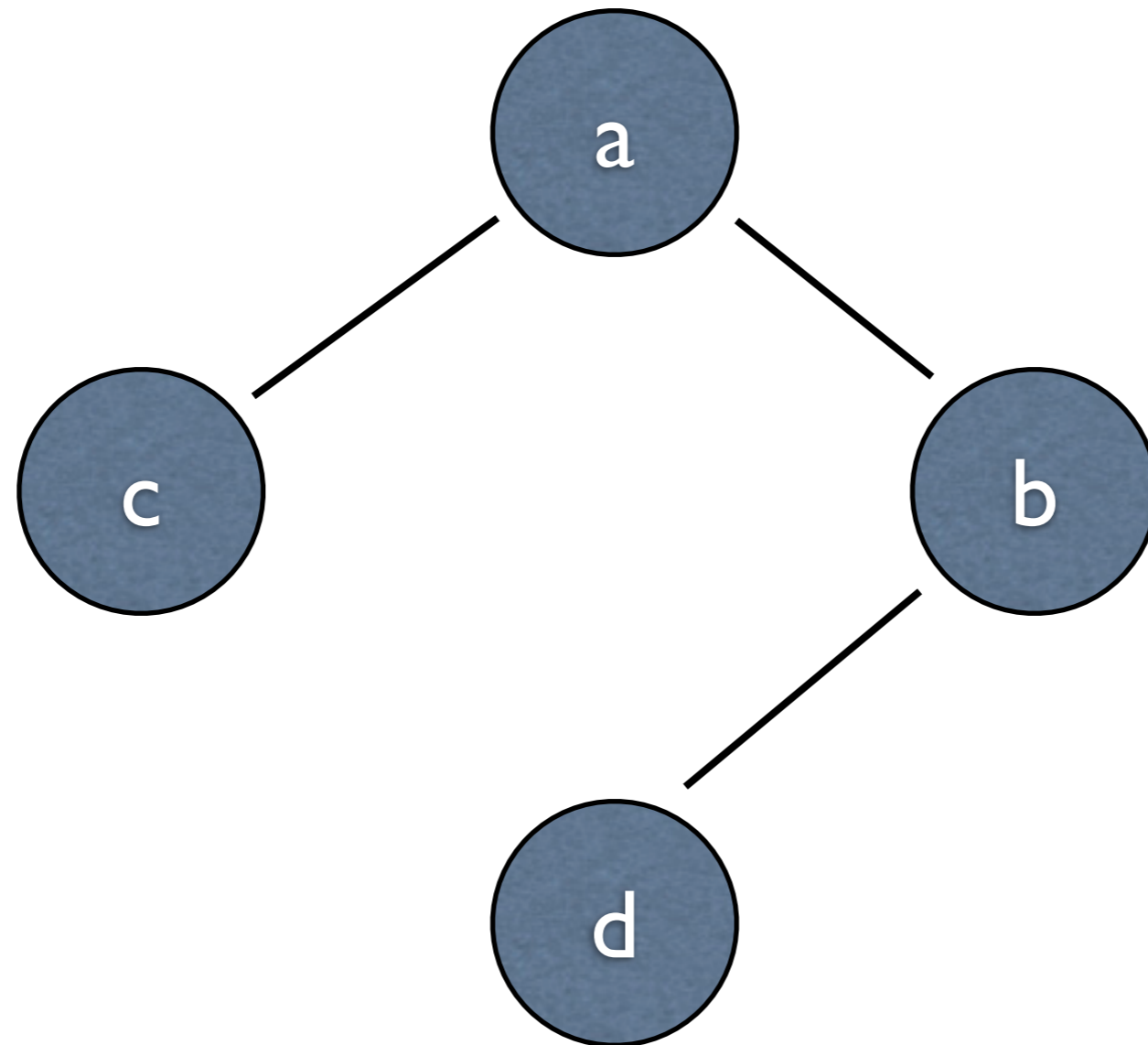
Übung



Graphendurchlauf

- Tiefensuche (depth first search, DFS)
- Breitensuche (breadth first search, BFS)
- Expansion $X(v)$ eines Graphen G
 - Der Baum, der entsteht, wenn man alle Pfade ausgehend von einem Knoten v durchläuft

Beispiel



Tiefendurchlauf

- Tiefendurchlauf entspricht dann einem Preoder-Durchlauf der Expansion

Preorder

```
sub preorder($node) {  
    print $node->{$value};  
    if ($node->{left} ne null) {  
        preorder($node->{left});  
    }  
    if ($node->{right} ne null) {  
        preorder($node->{right});  
    }  
}
```

Inorder

```
sub inorder($node) {  
    if ($node->{left} ne null) {  
        preorder($node->{left});  
    }  
    print $node->{$value};  
    if ($node->{right} ne null) {  
        preorder($node->{right});  
    }  
}
```

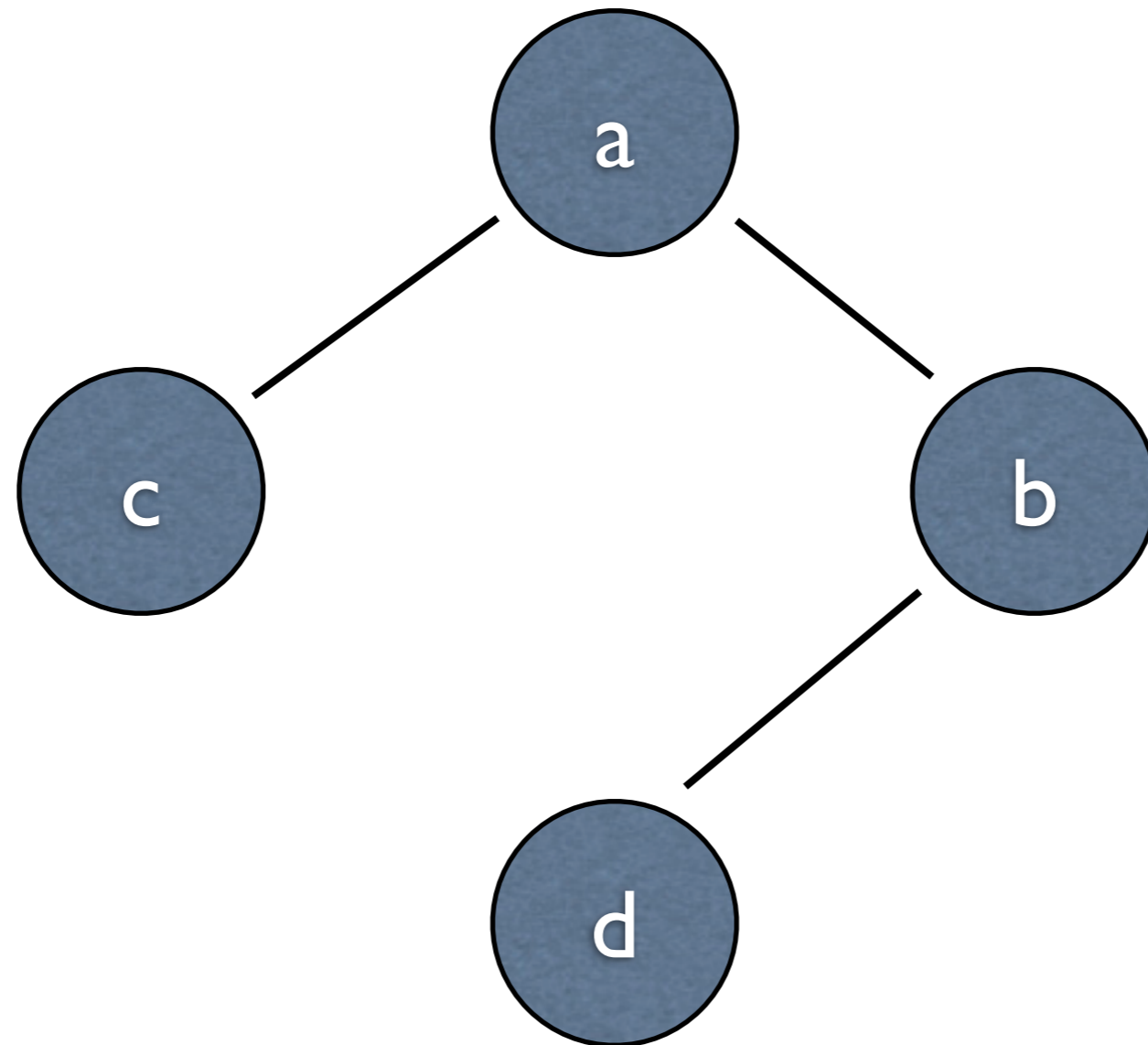
Postorder

```
sub postorder($node) {  
    if ($node->{left} ne null) {  
        preorder($node->{left});  
    }  
    if ($node->{right} ne null) {  
        preorder($node->{right});  
    }  
    print $node->{$value};  
}
```

Breitendurchlauf

- Expansion analog zu Tiefendurchlauf, allerdings werden zunächst alle Knoten auf einer Ebene besucht, dann erst eine Ebene tiefer gegangen

Beispiel



Edsger Wybe Dijkstra

- niederländischer Informatiker (11. Mai 1930 – 06. August 2002)
- Entdecker des gleichnamigen Algorithmus
- ‚Go To Statement Considered Harmful‘
- ‚2 or more, use for.‘

Edsger Wybe Dijkstra

- ,Computer science is as much about computers as astronomy is about telescopes.'
- ,It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.'

Edsger Wybe Dijkstra

- ,The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.'

Dijkstra's Algorithm

- auch: Single Source Shortest Paths
- findet für einen gegebenen Anfangsknoten die kürzesten Distanzen zu allen anderen Knoten im Graph

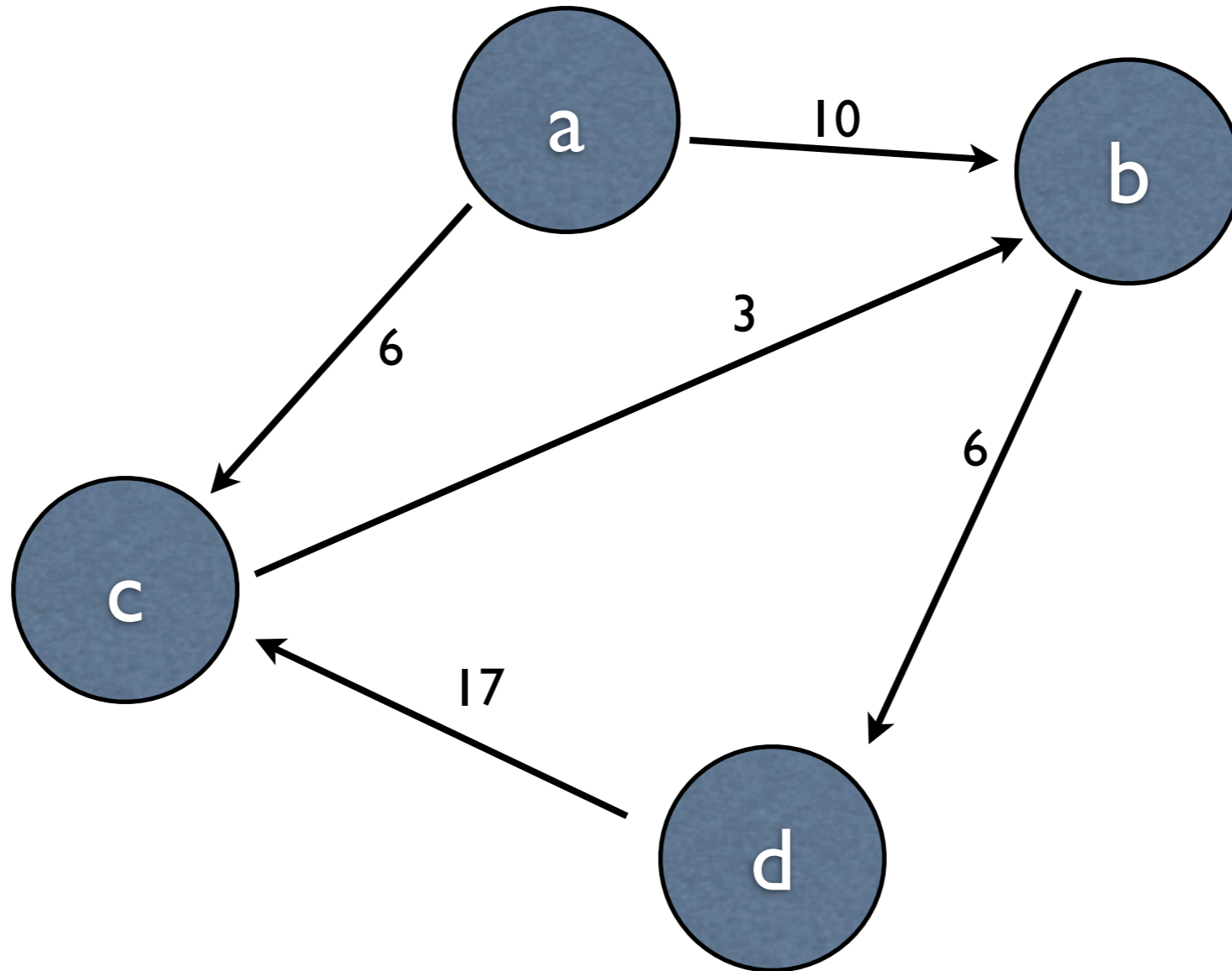
Dijkstra's Algorithm

- Idee: In jedem Durchlauf wird für einen Knoten, der ‚billigste‘ Weg von der Quelle gesucht. Nachzulesen in [1].
- Menge der bereits erfassten Knoten $M \subseteq V$
- $\text{costs}[i, j]$ = Kosten von i nach j
- $\text{low}[i]$ = geringste Kosten von Quelle zu i

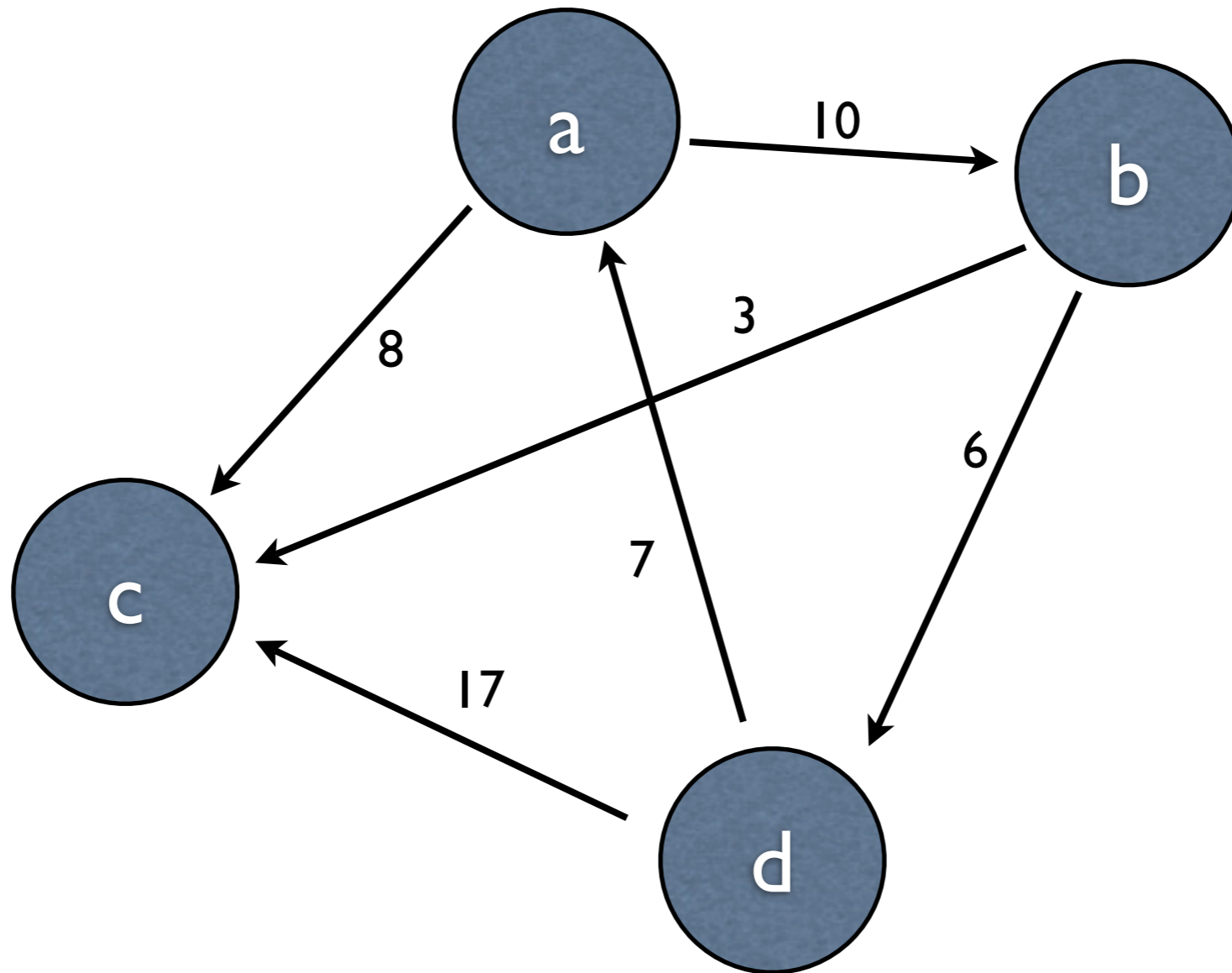
Dijkstra's Algorithm

```
M = {1}
low[1] = 0;
foreach (node > 1)
    low[node] = costs[1, i];
for (i = 1; i < number_of_nodes; i++) {
    add node with smallest value of low[i] to M; // via priority queue / heap
    foreach (node NOT IN M) {
        low[node] = min(low[node], low[i] + costs[i, node]);
    }
}
```

Übung



Übung



Dijkstra's Algorithm

- $M = \{a\}$, $\text{low}[a] = 0$, $\text{low}[b] = 10$, $\text{low}[c] = 6$, $\text{low}[d] = \infty$
- $M = \{a, c\}$, $\text{low}[a] = 0$, $\text{low}[b] = 9$, $\text{low}[c] = 6$, $\text{low}[d] = \infty$
- $M = \{a, c, b\}$, $\text{low}[a] = 0$, $\text{low}[b] = 9$, $\text{low}[c] = 8$, $\text{low}[d] = 15$

Dijkstra's Algorithm

- $M = \{a, c, b, d\}$, $\text{low}[a] = 0$, $\text{low}[b] = 9$, $\text{low}[c] = 8$, $\text{low}[d] = 15$

Dijkstra's Algorithm

- siehe auch
- http://en.wikipedia.org/wiki/Dijkstra's_algorithm
- http://students.ceid.upatras.gr/~papagel/project/kef5_7_1.htm
- <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/119466>

Priority Queue / Heap

- <http://search.cpan.org/~jmm/Heap-0.80/lib/Heap.pm>
- (sudo) perl -MCPAN -e shell
- install Heap

Zurück zu WordNet

- mögliche Anwendungen
 - Query Anreicherung im Information Retrieval
 - als Thesaurus
 - semantische Ähnlichkeit zwischen Worten: <http://search.cpan.org/~jasonm/Text-Similarity-0.02/lib/Text/Similarity.pm>

Zurück zu WordNet

- Visualisierung
 - Relationen in WordNet lassen sich graphisch, z.B. als SVG (Scalable Vector Graphics, ein XML-Format für Vektorgrafik) darstellen
 - Umsetzung in XML, RDF

Graph::Easy

- `(sudo) perl -MCPAN -e shell`
- `install Graph::Easy`
- `install Graph::Easy::As_svg`

WordNet::QueryData

```
use Data::Dumper;
use Graph::Easy;
use WordNet::QueryData;

my $wn = WordNet::QueryData->new();
my $graph = Graph::Easy->new();
my $adjacency_matrix;
get_related_terms($ARGV[0]);

foreach my $row_index (keys(%{$adjacency_matrix})) {
    $graph->add_node($row_index);
    foreach my $column_index (keys(%{$adjacency_matrix->{$row_index}})) {
        $graph->add_node($column_index);
        $graph->add_edge_once($row_index, $column_index);
    }
}

$graph->{timeout} = 20;

print $graph->as_svg();
```

WordNet::QueryData

```
sub get_related_terms() {  
  my $term = shift;  
  my $i, $j = 0;  
  
  foreach my $related_term ($wn->querySense($term, "hypos")) {  
    $adjacency_matrix->{$term}->{$related_term} = 1;  
    get_related_terms($related_term);  
  }  
}
```

Aufgabe a)

- schreiben Sie ein Modul, das den Algorithmus von Dijkstra implementiert
- das Modul soll einen Graphen (repräsentiert durch einen Hash-of-Hashes) und einen String als Identifikator für den Startknoten entgegennehmen und für jeden Knoten den Abstand zum Startknoten zurückgeben

Aufgabe b)

- schreiben Sie ein Modul, das einen String entgegennimmt und für jedes Wort verwandte Worte (Hyponyme und Hypernyme) vorschlägt
- evtl. ist dazu ein rekursiver Aufruf der entsprechenden Methode notwendig, falls z.B. erst die Wortarten aufgelöst werden müssen (z.B. cat#n#l)

Aufgabe b)

- gestalten Sie die Methode derart, dass Zwischenstufen wie `cat#n#l` erkannt werden und die Methode mit diesem String als Argument noch einmal rekursiv aufgerufen wird
- bereiten Sie die Ergebnisse fürs menschliche Auge auf (sprich: Entfernen Sie etwaige `#`, POS Tags und Indizes)

Aufgabe

- Abgabe der Aufgabe in Teams per E-Mail
- Abgabe bis zum 17.05.2007 0 Uhr

Referenzen

- [1] Güting, R. H. u. Dieker, S. (2004):
Datenstrukturen und Algorithmen, Teubner
Verlag, Wiesbaden.

**Vielen Dank für Ihre
Aufmerksamkeit!**