

Programmierpraktikum

Sommersemester 2007 - 25.05.2007

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

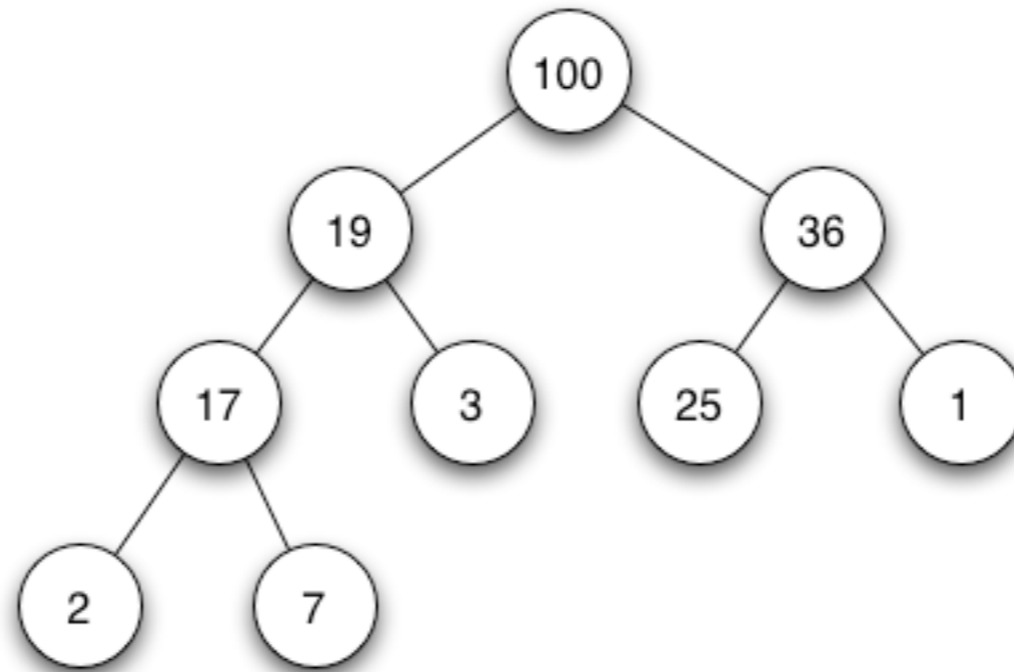
Heute

- Datenstrukturen
 - Heap
 - Binomial Heap
 - Fibonacci Heap
- Anwendungen für XML-kodierte Datenstrukturen

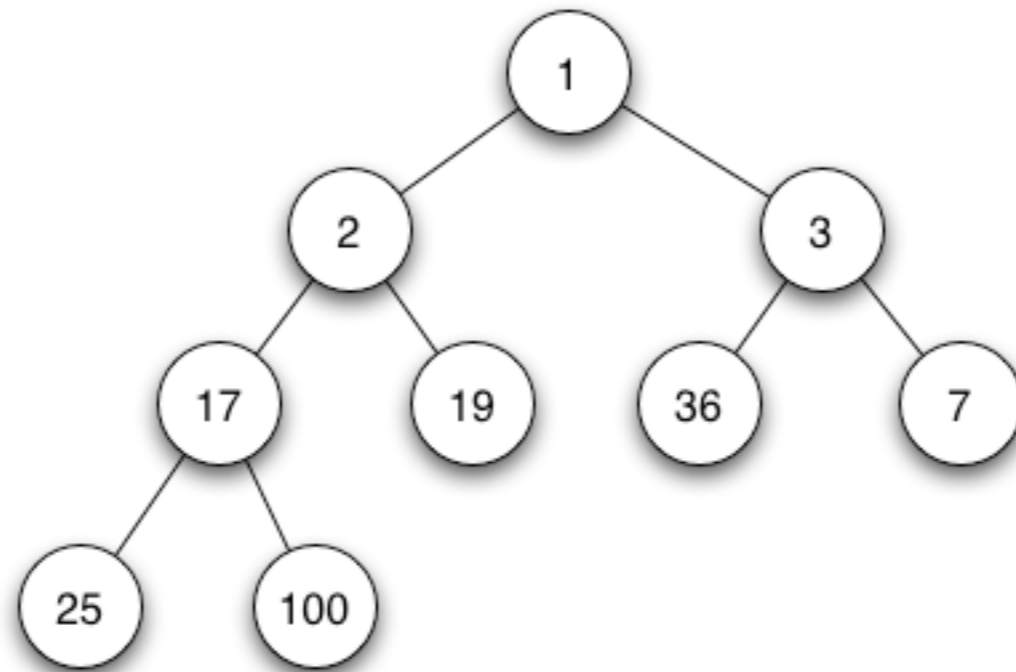
Heap

- auch: priority queue
- binärer Baum
- jeder Kindknoten ist kleiner als sein Mutterknoten (Max Heap)
- jeder Kindknoten ist größer als sein Mutterknoten (Min Heap)
- das gesuchte Element ist immer oben

Max Heap



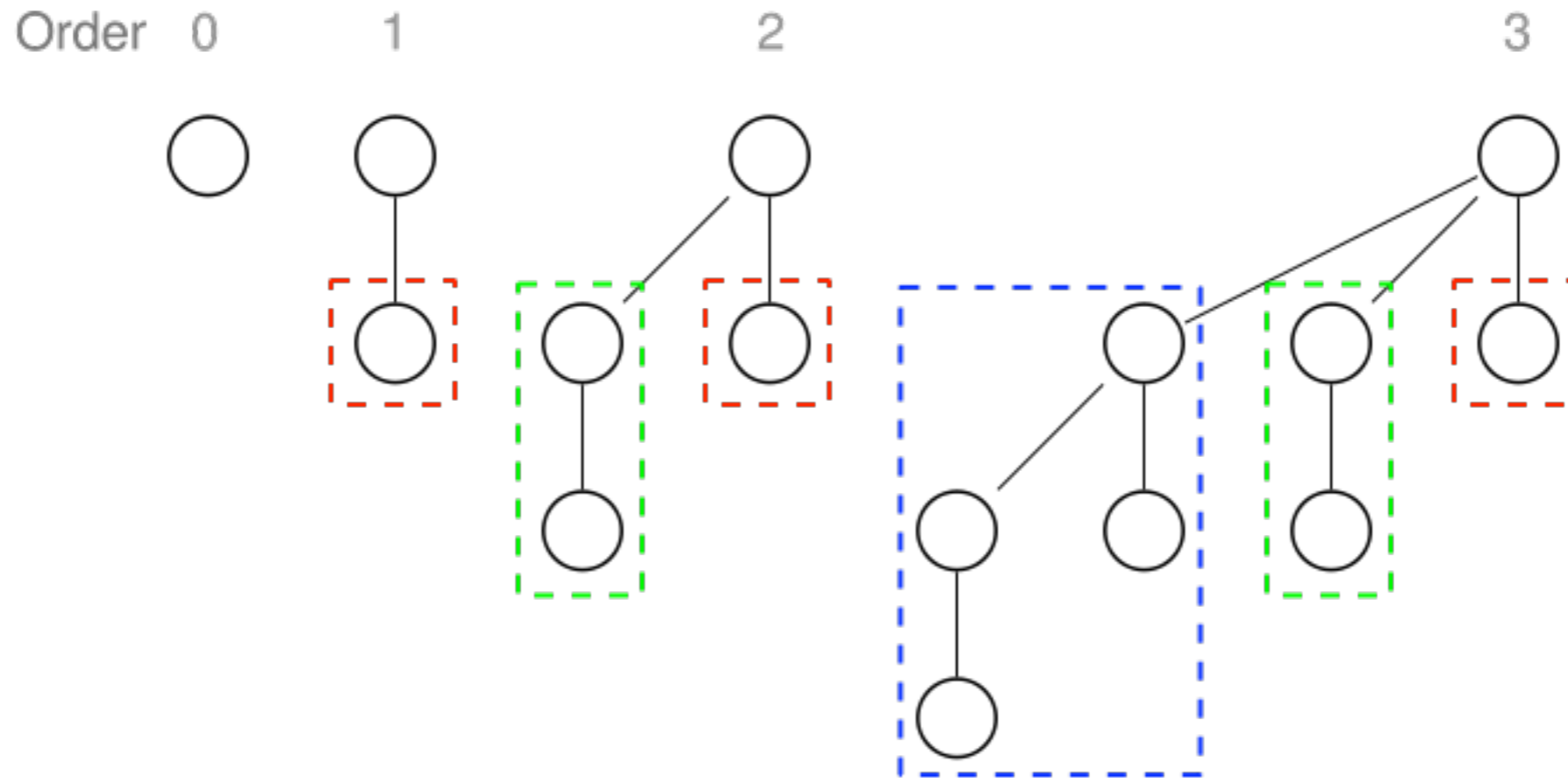
Min Heap



Binomial Heap

- Mischwald / Merge-Find-Set
- besteht aus einer Menge von Bäumen
- rekursiv definiert: Ordnung 0: 1 Knoten,
Ordnung k : Wurzel hat k Kinder und
Kinder sind vom Grad $k-1, k-2, \dots, 1, 0$

Binomial Heap



Fibonacci Zahlen

- 1, 1, 2, 3, 5, 8, 13...
- rekursiv definiert als $k_n = k_{n-1} + k_{n-2}$ für $n > 1$ und $k_0 = 0, k_1 = k_2 = 1$
- Ursprung: Pingala (ca. 450 bis 200 BCE)
bzw. Leonardo von Pisa (Fibonacci) im Jahre 1202

Fibonacci Zahlen

- eng verbunden mit dem Goldenen Schnitt
- $F(n) = (\varphi^n - (1 - \varphi)^n) / \sqrt{5}$

Goldener Schnitt

- irrationale Zahl
- $\varphi = (1 + \sqrt{5}) / 2$
- teilt ein Intervall so, dass der größere Teil und der kleinere Teil im selben Verhältnis stehen wie das Intervall und der größere Teil

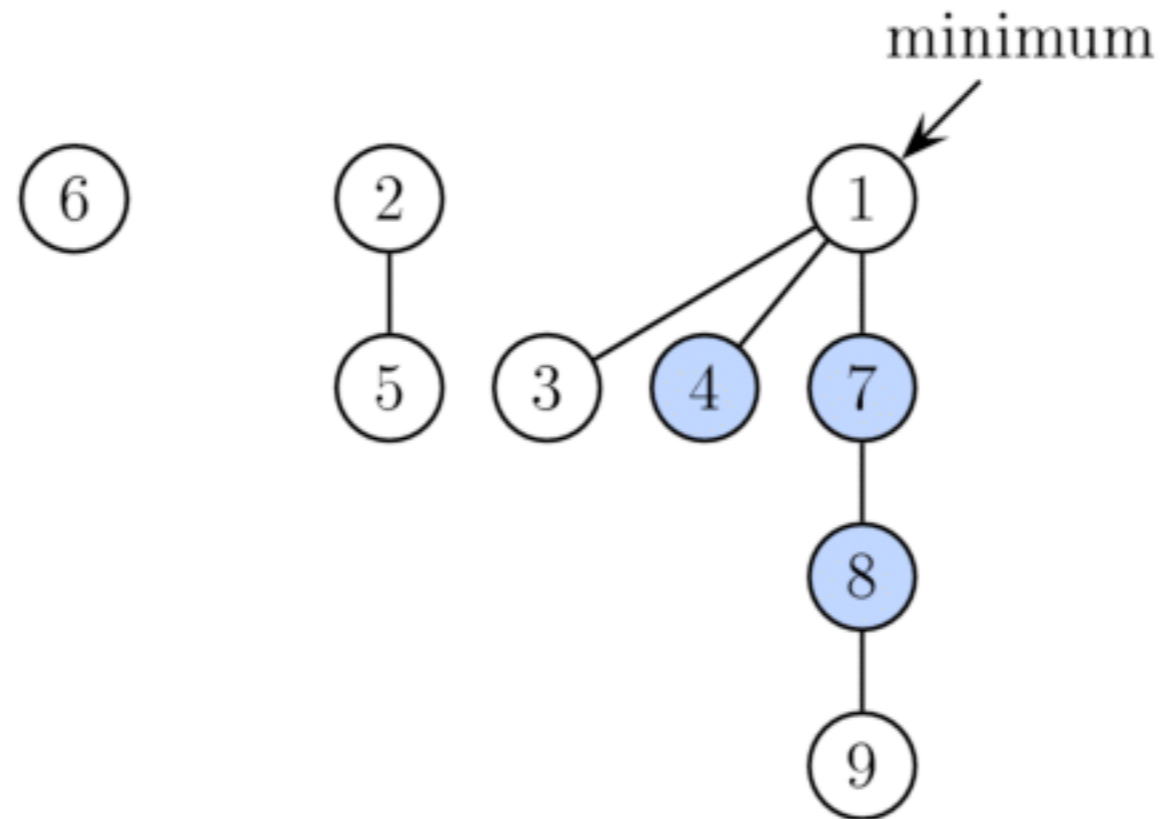
Goldener Schnitt

- Anwendung z.B. auch bei ‚guten‘ Hashfunktionen (Gleichmäßige Verteilung von Elementen)
- Anwendung neben Mathematik und Informatik in der Architektur, Kunst, Musik... auch viele natürliche Strukturen (Blätter etc.) scheinen nach dem Goldenen Schnitt ‚konstruiert‘

Fibonacci Heap

- ähnlich dem Binomial Heap
- aber keine vorgegebene Struktur
- im Extremfall ist jedes Element ein eigener Baum
- Name rührt von der minimalen Größe F_{k+2} ($k+2$ te Fibonacci-Zahl) von Unterbäumen einer Wurzel vom Grad k

Fibonacci Heap



XML-Anwendungen

- XML Dateien einlesen
- XML Dateien ausgeben
- XML Datentypen in andere Datentypen transformieren
- XML Datentypen einlesen und weiterverarbeiten

Anwendungen

- RSS Newsfeeds aggregieren (Quellen zusammenführen und filtern)
- RSS (Really Simple Syndication, Rich Site Summary, RDF Site Summary) ist ein RDF-Dialekt zur semantischen Anreicherung von häufig aktualisierten Inhalten (Nachrichten, Blogs)

XML::RSS

- dient der Verarbeitung von RSS-Code
- XML::RSS parst RSS-Code weitestgehend selbständig
- kapselt die Details, der eigentliche XML-Code wird versteckt
- der Entwickler muss sich lediglich mit Perl-Strukturen auseinandersetzen

XML::RSS

- `(sudo) perl -MCPAN -e shell`
- `install XML::RSS`

XML::RSS::Aggregate

- dient der Aggregation (Vereinigung von RSS- Quellen)
- führt verschiedene RSS-Quellen (Newsfeeds) zu einer neuen Struktur zusammen

XML::RSS::Aggregate

- (sudo) perl -MCPAN -e shell
- install XML::RSS::Aggregate

XML::RSS::Aggregate

```
# use RSS parser
use XML::RSS;

# use aggregator
use XML::RSS::Aggregate;

# initialise aggregator
my $rss = new XML::RSS::Aggregate(
    title    => 'Test',
    link     => 'http://www.test.com/',
    sources => \@urls,
);

# aggregate
$rss->aggregate();

# iterate over aggregated RSS items
foreach my $item (@{$rss->{items}}) {
    print Dumper $item;
}
```

Anwendungen

- unter URLs abgelegte Daten abrufen
- HTML Code einlesen
- textuelle Inhalte weiter verarbeiten
- durch Markup-Code markierte Metainformationen verarbeiten

LWP

- libwww
- World Wide Web Library für Perl
- stellt Klassen für HTTP Requests und Responses zur Verfügung

LWP::Request

- dient dem Absetzen von HTTP Requests
- HTTP Requests sind Ressourcen-Anforderungen eines Clients an einen Server
- REST (Representation State Transfer)
- idempotente Methoden: GET, PUT, DELETE
- POST

LWP::Response

- modelliert HTTP Responses
- HTTP Response: Antwort eines Servers auf eine Client-Anfrage
- Status Codes: http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

LWP::UserAgent

- simuliert eine Browserumgebung (sog. UserAgent)
- dient dem Senden von Requests und dem Empfangen von Responses
- erlaubt zusätzlich das Modifizieren des UserAgent Strings oder des HTTP Headers
- LWP::RobotUA für Robot Anwendungen

LWP

- `(sudo) perl -MCPAN -e shell`
- `install LWP`
- `install LWP::UserAgent`
- `evtl. install HTTP::Request` und `install HTTP::Response`

LWP

```
# create user agent
use LWP::UserAgent;

# create user agent
my $userAgent = LWP::UserAgent->new();
$userAgent->agent(
    "'Mozilla/5.0 (compatible)'"
);

# request
my $request = HTTP::Request->new(GET => $url);

# pass request to the user agent
my $response = $userAgent->request($request);

# check the response
if ($response->is_success) {
    print $response->content;
} else {
    print $response->status_line . "\n";
}
```

HTML::Parser

- dient dem Parsen von HTML-Dokumenten
- bedient sich des in der letzten Sitzung vorgestellten Event Handler Prinzips
- <http://search.cpan.org/~gaas/HTML-Parser-3.56/Parser.pm>

HTML::Parser

- `(sudo) perl -MCPAN -e shell`
- `install HTML::Parser`

HTML::Parser

```
# use packages
use HTML::Entities;
use HTML::Parser;

# initialise HTML parser
$parser = HTML::Parser->new(
    api_version => 3,
    handlers => {
        start => [\@tags, "tagname"],
        text => [\@content, "text"]
    }
);

# parse using HTML entity decoder
$parser->parse(
    HTML::Entities::decode_entities(
        $response->content
    )
);
```

HTML::Parser

```
# use packages
use HTML::Entities;
use HTML::Parser;

# initialise HTML parser
$parser = HTML::Parser->new(
    api_version => 3,
    handlers    => {
        start => [\&startHandler, "tagname,attr,self"]
    }
);

# start handler
sub startHandler(
    ...
);
```

Vektorraummodelle

- jeder String lässt sich als Vektor der in dem String enthaltenen Terme darstellen
- als Indices fungieren dabei die Terme selbst,
- als Wert fungiert die Häufigkeit des jeweiligen Vorkommens eines Terms im String

Vektorraummodelle

- Beispiel: ‚Dieser Test ist ein Test‘
- \$vector = (Dieser => 1, Test => 2, ist => 1, ein => 1)
- n.b.: Die Information über die Position der einzelnen Token geht dabei verloren

Vektorraummodelle

- durch eine Repräsentation als Vektoren lassen sich zwei Strings anhand des Winkels zwischen ihren Vektoren auf Ähnlichkeit hin überprüfen
- dies wird z.B. bei Suchmaschinen genutzt, um auf Anfragen passende Dokumente zurückzuliefern

Kosinus-Ähnlichkeit

- engl. Cosine Similarity
- $\cos \alpha = \langle \vec{x}, \vec{y} \rangle / (|\vec{x}| |\vec{y}|)$
- je mehr $\cos \alpha$ Richtung 1 konvergiert, desto ähnlicher sind die Vektoren, da $\cos(0) = 1$ bedeutet, dass die Vektoren aufeinander liegen

Aufgabe a)

- Erstellen Sie ein Modul, das
 - XML::RSS und XML::RSS::Aggregate nutzt
 - Feed-URLs zeilenweise aus einer Textdatei (Beispiel kommt per E-Mail) einliest
 - die Feeds aggregiert

Aufgabe a)

- Erstellen Sie ein Modul, das
 - die 5 am besten zu einem oder mehreren per Kommandozeile gegebenen Suchtermen passenden Artikel (Link, Titel, Beschreibung, falls vorhanden) ausgibt
- Tipp: Bei evtl. Ausgabefehlern versuchen Sie `use bytes;` zu benutzen

Aufgabe b)

- Erstellen Sie ein Modul, das
 - ein HTML-Dokument von einer URL holt
 - Plain Text extrahiert
 - Types und Token zählt und deren jeweilige Gesamtzahl ausgibt
 - Links (`<a href ...>...`) extrahiert und deren Titel und Ziel ausgibt

Aufgabe b)

- für jedes Dokument einen Term-Vektor mit den jeweiligen Typen als Schlüssel und den jeweiligen Häufigkeiten als Werten erstellt

Aufgabe

- Newsfeeds
 - <http://www.rssfeeds.com/>
 - <http://www.rss-verzeichnis.de/>
- Abgabe der Aufgabe in Teams per E-Mail
- Abgabe bis zum 06.06.2007, 0 Uhr

Nächste Sitzung

- Am 08.06.2007 (in der ersten Juniwoche sind Pfingsferien!)

Referenzen

- Güting, R. H. u. Dieker, S. (2004):
Datenstrukturen und Algorithmen, Teubner
Verlag, Wiesbaden.
- http://en.wikipedia.org/wiki/Binary_heap
- http://en.wikipedia.org/wiki/Binomial_heap
- http://en.wikipedia.org/wiki/Fibonacci_heap

**Vielen Dank für Ihre
Aufmerksamkeit!**