

ADS: Algorithmen und Datenstrukturen

Teil VI

Peter F. Stadler & Konstantin Klemm

Bioinformatics Group, Dept. of Computer Science & Interdisciplinary Center for
Bioinformatics, **University of Leipzig**

01. Dezember 2010

Merge-Sort

- Anwendbar für internes und externes Sortieren
- Ansatz (Divide-and-Conquer-Strategie)
 - rekursives Zerlegen in jeweils gleich große Teilfolgen
 - Verschmelzen (Mischen) der sortierten Teilfolgen

Algorithmus (Rekursives Merge-Sort)

```
sort(A,l,r) sortiert das Feld A im Bereich l bis r
  Falls r-l=1: fertig
  Falls r-l>1
    Mitte der Liste bestimmen: int m = (l+r+1)/2
    linken Teil sortieren: sort(A,l,m-1)
    rechten Teil sortieren: sort(A,m,r)
    zusammenfügen: merge(A,l,m,r)
```

Mischen für Merge-Sort

Pro Teilliste wird Zeiger (Index) auf nächstes Element geführt.

- jeweils kleinstes Element wird in Ergebnisliste übernommen und Indexzeiger fortgeschaltet
- sobald eine Teilliste “erschöpft” ist, werden alle verbleibenden Elemente der anderen Teilliste in die Ergebnisliste übernommen
- lineare Kosten

```
merge(A,l,m,r)
```

```
// 1. Schritt: Schleife zum Füllen von B:
```

```
j=l; k=m;
```

```
for (i=0; i<r-l+1; i++)
```

```
    if (k>r or (j<m and A[j]<=A[k])) B[i] = A[j++];
```

```
    else B[i] = A[k++];
```

```
// 2. Schritt: Rueckkopieren
```

```
for (i=0, i< r-l+1); i++) A[l+i] = B[i];
```

Merge-Sort, nicht-rekursiv

Algorithmus

- zunächst werden benachbarte Elemente zu sortierten Teillisten der Länge 2 gemischt
- fortgesetztes Mischen benachbarter (sortierter) Teillisten
- Länge sortierter Teillisten verdoppelt sich pro Durchgang
- Sortierung ist beendet, sobald in einem Durchgang nur noch zwei Teillisten verschmolzen werden.

Beispiel: Merge-Sort

5	3	4	6	8	1	2	7							
5		3		4		6		8		1		2		7
3	5		4	6		1	8		2	7				
3	4	5	6		1	2	7	8						
1	2	3	4	5	6	7	8							

Beispiel: Merge-Sort

5 3 4 6 8 1 2 7

5 | 3 | 4 | 6 | 8 | 1 | 2 | 7

3 5 | 4 6 | 1 8 | 2 7

3 4 5 6 | 1 2 7 8

1 2 3 4 5 6 7 8

Beispiel: Merge-Sort

5 3 4 6 8 1 2 7

5 | 3 | 4 | 6 | 8 | 1 | 2 | 7

3 5 | 4 6 | 1 8 | 2 7

3 4 5 6 | 1 2 7 8

1 2 3 4 5 6 7 8

Beispiel: Merge-Sort

5 3 4 6 8 1 2 7

5 | 3 | 4 | 6 | 8 | 1 | 2 | 7

3 5 | 4 6 | 1 8 | 2 7

3 4 5 6 | 1 2 7 8

1 2 3 4 5 6 7 8

Beispiel: Merge-Sort

```
5  3  4  6  8  1  2  7
5 | 3 | 4 | 6 | 8 | 1 | 2 | 7
3  5 | 4  6 | 1  8 | 2  7
3  4  5  6 | 1  2  7  8
1  2  3  4  5  6  7  8
```

Natürliches Merge-Sort

Algorithmus

- statt mit 1-elementigen Teillisten zu beginnen, werden bereits anfangs möglichst lange sortierte Teilfolgen (“runs”) verwendet
- Nutzung einer bereits vorliegenden (natürlichen) Sortierung in der Eingabefolge

Beispiel: Natürliches Merge-Sort

5	3	4	6	8	1	2	7		
5		3	4	6	8		1	2	7
3	4	5	6	8		1	2	7	
1	2	3	4	5	6	7	8		

Beispiel: Natürliches Merge-Sort

5	3	4	6	8	1	2	7		
5		3	4	6	8		1	2	7
3	4	5	6	8		1	2	7	
1	2	3	4	5	6	7	8		

Beispiel: Natürliches Merge-Sort

5	3	4	6	8	1	2	7		
5		3	4	6	8		1	2	7
3	4	5	6	8		1	2	7	
1	2	3	4	5	6	7	8		

Beispiel: Natürliches Merge-Sort

5	3	4	6	8	1	2	7		
5		3	4	6	8		1	2	7
3	4	5	6	8		1	2	7	
1	2	3	4	5	6	7	8		

Zusammenfassung: Sortieren

- Kosten allgemeiner Sortierverfahren wenigstens $O(n \log n)$
- Elementare Sortierverfahren: Kosten $O(n^2)$
 - einfache Implementierung; ausreichend bei kleinem n
 - gute Lösung: Insertion Sort
- $O(n \log n)$ -Verfahren: Heap-Sort, Quick-Sort, Merge-Sort
 - Heap-Sort und Merge-Sort sind worst-case-optimal, $O(n \log n)$
 - in Messungen erzielte Quick-Sort in den meisten Fällen die besten Ergebnisse
- Begrenzung der Kosten für Umkopieren durch indirekte Sortierverfahren
- Generall vorteilhafte Eigenschaften von Sortierverfahren
 - Ausnutzen einer weitgehenden Vorsortierung
 - Stabilität

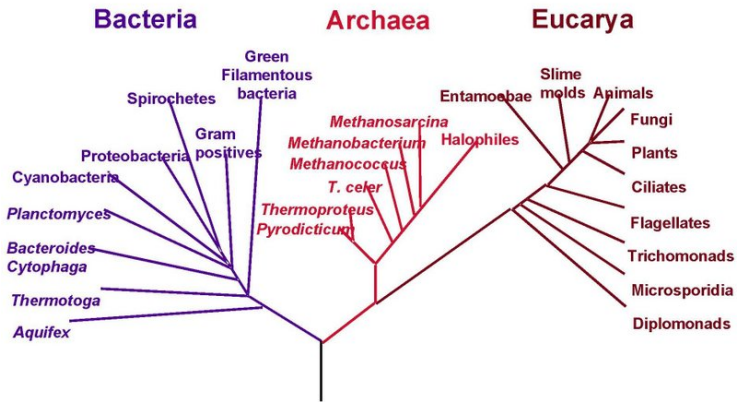
Bäume

- Ein *Graph* ist ein Paar (V, E) . Hierbei ist V eine Menge (=Knotenmenge) und E eine Menge von ungeordneten Paaren (=Kanten) in V .
- Ein *Wald* ist ein Graph ohne Zyklen.
- Ein *Baum* ist ein zusammenhängender Wald.

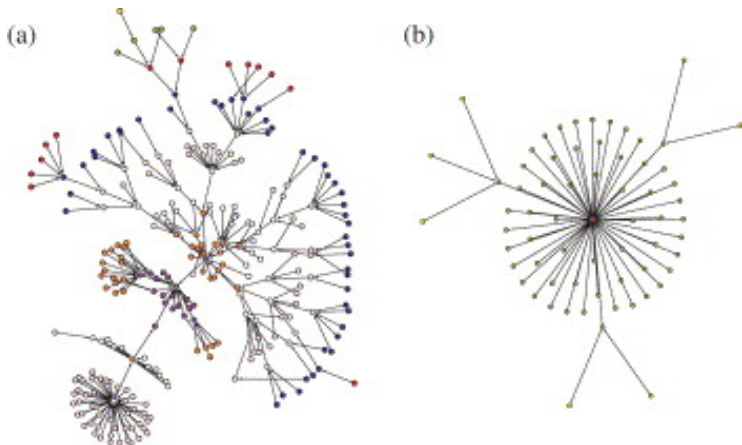
hier noch nicht definiert, aber intuitiv klar: Zyklus, Zusammenhang.
Beispiele → Tafel.

Beispiel 1: Phylogenetischer Baum

Phylogenetic Tree of Life



Beispiel 2: Verzeichnisbäume



Klemm et al., *Analysis of attachment models for directory and file trees*, Physica D (2006)

Orientierte Bäume

Eine Menge B ist ein orientierter (Wurzel-) Baum, falls

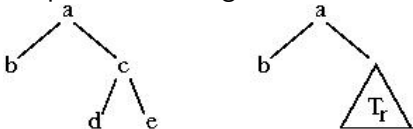
- 1 in B ein ausgezeichnetes Element w – die Wurzel von B – existiert
- 2 die Elemente in $B \setminus \{w\}$ disjunkt zerlegt werden können in B_1, B_2, \dots, B_m , wobei jedes B_i ebenfalls ein orientierter Baum ist.

Anschaulich:

- Die Kanten des Graphen sind gerichtet.
- Von der Wurzel gehen nur Kanten aus, keine treffen ein.
- Jeder Knoten ist von der Wurzel aus auf genau einem Weg entlang der gerichteten Kanten erreichbar.
- Für jeden Nicht-Wurzelknoten gibt es Knoten, die nicht entlang der gerichteten Kanten erreichbar sind.

Darstellungsarten für orientierte Bäume

1 Graphendarstellung



2 Mengendarstellung

$\{ \{a, b, c, d, e\}, \{b\}, \{c, d, e\}, \{d\}, \{e\} \}$

3 Klammerdarstellung

$(a, (b), (c, (d), (e)))$

4 Rekursives Einrücken

a
 b
 c
 d
 e

Binärbäume

Ein Binärbaum ist eine endliche Menge von Elementen, die entweder leer ist oder ein ausgezeichnetes Element - die Wurzel des Baumes - besitzt und folgende Eigenschaften aufweist:

- Die verbleibenden Elemente sind in zwei disjunkte Untermengen zerlegt.
- Jede Untermenge ist selbst wieder ein Binärbaum und heißt linker bzw. rechter Unterbaum des ursprünglichen Baumes

Formale ADT-Spezifikation: BINTREE

Datentyp BINTREE, Basistyp ELEM

Operationen:

CREATE:		→ BINTREE
EMPTY:	BINTREE	→ {TRUE, FALSE}
BUILD:	BINTREE × ELEM × BINTREE	→ BINTREE
LEFT:	BINTREE \ {0}	→ BINTREE
ROOT:	BINTREE \ {0}	→ ELEM
RIGHT:	BINTREE \ {0}	→ BINTREE

Axiome

- 1 CREATE() = 0;
- 2 EMPTY (CREATE()) = TRUE;
- 3 $\forall l, r \in \text{BINTREE}, \forall d \in \text{ELEM}$:
EMPTY (BUILD (l, d, r)) = FALSE
LEFT (BUILD (l, d, r)) = l;
ROOT (BUILD (l, d, r)) = d;
RIGHT (BUILD (l, d, r)) = r;

Beispiele für Konstruktion

Welche Binärbäume entstehen durch

- BUILD (BUILD (0, b , BUILD (0, d , 0)), a , BUILD (0, c , 0))
- BUILD (BUILD (BUILD (0, d , 0), b , 0), a , BUILD (0, c , 0))

?

Eigenschaften von Binärbäumen I

Satz: Die maximale Anzahl von Knoten eines Binärbaumes

- 1 auf Stufe i ist 2^i , $i \geq 0$
- 2 der Höhe h ist $2^h - 1$, $h \geq 0$ (Der leere Baum hat Höhe 0)

Definition: Ein *vollständiger* Binärbaum der Höhe $k + 1$ hat folgende Eigenschaften:

- Jeder Knoten der Stufe k ist ein Blatt.
- Jeder Knoten auf einer Stufe $< k$ hat nicht-leere linke und rechte Unterbäume.

Definition: In einem *strikten* Binärbaum besitzt jeder innere Knoten nicht-leere linke und rechte Unterbäume

Eigenschaften von Binärbäumen II

Definition: Ein *ausgeglichener* Binärbaum ist ein Binärbaum, so dass gilt:

- 1 Jedes Blatt im Baum ist auf Stufe k oder $k + 1$, ($k \geq 0$).
- 2 Jeder Knoten auf Stufe $< k$ hat nicht-leere linke und rechte Teilbäume

Definition: Ein *fast vollständiger* Binärbaum ist ein ausgeglichener Binärbaum, bei dem zusätzlich gilt:

Falls ein innerer Knoten einen rechten Nachfolger auf Stufe $k + 1$ besitzt, dann ist sein linker Teilbaum vollständig mit Blättern auf Stufe $k + 1$.