11. AuD Tafelübung T-C3

Simon Ruderich

19. Januar 2011

$\mathcal{O}$-Kalkül

Funktionsaufwände in Schleifen

$f(j) \in \mathcal{O}(j^3)$

```
for (i = 0; i < x; i++) {
    f(i);
}
```

$$\sum_{i=0}^{x} i^3 = \frac{1}{4}x^2(x+1)^2 \in \mathcal{O}(x^4)$$

$\mathcal{O}$-Kalkül

Funktionsaufwände in Schleifen

$f(j) \in \mathcal{O}(j^2)$

```
for (i = 1; i < x; i *= 2) {
    f(i);
}

for (k = 0; k < ld(x); k++) {
    f(2^k);
}
```

$$\sum_{k=0}^{\mathrm{ld}(x)} \left(2^k\right)^2 = \frac{1}{3}(4x^2 - 1) \in \mathcal{O}(x^2)$$

Verkettete Liste

Verkettete Liste

- Datenstruktur zum Verwalten von Daten
- jeder Eintrag hat einen Verweis auf das nächste Element

Vorteile

- flexible Größe, keine Speicherverschwendung
- Einfügen/Löschen am Anfang/in der Mitte einfach

Nachteile

- kein wahlfreier Zugriff (Aufwand: $\mathcal{O}(n)$)
- schwerer zu implementieren (als z. B. Array)

Verkettete Liste

Verkettete Listen

jedes Element hat einen Zeiger auf das ...

einfach verkettete Liste	...nächste Elemente
doppelt verkettete Liste	...nächste und vorherige Element
zirkuläre Listen	letztes Element zeigt auf das Erste
Wächterelement	spezieller Listenkopf der immer in der Liste ist macht ggf. Sonderfälle einfacher

Verkettete Liste

Operationen

- Einfügen eines Elements:
  - am Anfang
  - in der Mitte
  - am Ende
- Löschen eines Elements: vgl. Einfügen
- Durchlaufen der Liste

Sonderfälle

- erstes Element einfügen/löschen
- leere Liste

Wäscheleine

Wäscheleine

- *zirkuläre*, doppelt-verkettete Liste
- `implements WaescheleineInterface`!
- Elemente der Liste sind `Kleiderhaken`
- API *genau* lesen
- Exceptions passend werfen
- nur ein Element in der Liste:
  next- und prev-Zeiger zeigen auf das Element selbst
- Element-Vergleich mit `equals()`, nicht mit `==`

Generische Klassenparameter

Generische Klassenparameter

- ermöglichen Typsicherheit zur Compile-Zeit
- z. B. eine Liste nur für bestimme Objekte

Deklaration

```
public class Liste<T> {
    void add(T element) { /* ... */ }
}
```

Instantiierung

```
Liste<Integer> x = new Liste<Integer>();
x.add(new Integer(5)); // funktioniert
x.add(new Double(5)); // Compiler-Fehler
```

Sortierverfahren

Eigenschaften

- Laufzeit des Algorithmus (best-, average- und worst-case)
- Platzbedarf des Algorithmus (z. B. in $\mathcal{O}$-Notation)
  - in-place (in situ): kein zusätzlicher Speicherplatz nötig
  - out-of-place: zusätzlicher Speicherplatz nötig
- Stabilität: „gleichwertige“ Elemente bleiben in der selben Reihenfolge

Bubblesort

Bubblesort

- Vergleich des aktuellen Element mit dem nächsten
- Tauschen falls es größer (kleiner) ist
- größere (kleinere) Elemente „blubbern“ hoch

Eigenschaften

Laufzeit	$\mathcal{O}(n^2)$
Speicherplatz	in-place
Stabilität	stabil

Mergesort

Mergesort

1. halbiere die zu sortierende Liste
2. sortiere die beiden neuen Teillisten rekursiv
   - halbiere die Listen solange, bis die Liste nur noch ein Element hat
   - eine einelementige Liste ist bereits sortiert
3. verschmelze die beiden sortierten Teillisten

Eigenschaften

Laufzeit	$\mathcal{O}(n \log n)$
Speicherplatz	in-place (Listen), i. d. R. out-of-place (Arrays)
Stabilität	stabil

Kein wahlfreier Zugriff nötig, besonders für Listen geeignet.

Mergesort

Beispiel

	89	5	40	84	20	95	7	14
T	89	5	40	84	20 |	95	7	14
T	89	5 |	40	84 |	20	95 |	7	14
T	89 |	5 |	40 |	84 |	20 |	95 |	7 |	14
M	5	89 |	40	84 |	20	95 |	7	14
M	5	40	84	89 |	7	14	20	95
M	5	7	14	20	40	84	89	95

Radix-Sort

Radix-Sort

1. betrachte die letzte Stelle der Elemente (von rechts)
2. sortiere die Elemente nach der Stelle in Fächer
3. führe die Elemente in *sortierter Reihenfolge* zusammen
4. betrachte die vorletzte Stelle und beginne erneut

Eigenschaften

Laufzeit	$\mathcal{O}(k \cdot n)$, n Anzahl der Elemente, k die Anzahl Stellen des Schlüssels
Speicherplatz	$\mathcal{O}(k \cdot n)$
Stabilität	stabil

Alphabet und Struktur des Schlüssels muss bekannt sein
für Zahlen (ggf. Strings) gut geeignet

Radix-Sort

Beispiel

	4521	7450	1563	7503	5502	1790	1764
4. Stelle	7450	1790	4521	5502	1563	7503	1764
3. Stelle	5502	7503	4521	7450	1563	1764	1790
2. Stelle	7450	5502	7503	4521	1563	1764	1790
1. Stelle	1563	1764	1790	4521	5502	7450	7503

Weitere Sortierverfahren

Weitere Sortierverfahren

- Bucketsort
- Quicksort
- Heapsort

Fragen

Fragen?
Evaluation