Algorithmik

Sebastian Iwanowski
FH Wedel

1. Einführung in die formale Behandlung von Algorithmen

Algorithmik 1

1.1 Vergleich von grundlegenden Sortiertechniken

- Funktionsweise der Algorithmen
  PermutationSort, SelectionSort, Mergesort, Quicksort

  Beschreibung in Worten, graphische Visualisierung mit Arrays

- Laufzeitabschätzung für den schlechtesten Fall

  Aufstellen von Rekursionsgleichungen, explizite Auflösung

  Abschätzung mit O-Notation

- Ergebnisse:

PermutationSort:	$O(\exp(n))$
SelectionSort:	$O(n^2)$
Mergesort:	$O(n \log n)$
Quicksort:	$O(n^2)$

Referenzen zum Nacharbeiten:
Alt S. 4 - 7

Algorithmik 1

1.1 Vergleich von grundlegenden Sortiertechniken

Im Detail: Quicksort

- Quicksort (A, i, j):

  A ist ein Array aus n Elementen (a[1], …, a[n]).
  i,j sind Indizes zwischen 1 und n.
  Am Ende sind die Elemente zwischen a[i] und a[j] aufsteigend sortiert.

- Partition (A,i,k,j) → order:

  Am Ende ist A zwischen a[i] und a[j] umsortiert,
  sodass zunächst nur Elemente ≤ x := a[k] kommen, dann x und dann nur Elemente > x.
  Der Rückgabewert order ist die neue Position von x.

- Implementierung von Quicksort (Start mit Quicksort (A,1,n)):

```
if i < j
   then k := Zufallszahl zwischen i und j;
        dividingIndex := Partition (A,k);
        Quicksort (A, i, dividingIndex-1);
        Quicksort (A, dividingIndex+1,j);
```

Referenzen zum Nacharbeiten und Vertiefen:
Cormen Kap. 7.1 (ohne Zufallszahl)

Algorithmik 1

1.1 Vergleich von grundlegenden Sortiertechniken

Im Detail: Quicksort

- Partition (A,i,k,j) → order:

  Am Ende ist A zwischen a[i] und a[j] umsortiert,
  sodass zunächst nur Elemente ≤ x := a[k] kommen, dann x und dann nur Elemente > x.
  Der Rückgabewert order ist die neue Position von x.

- Implementierung von Partition:

```
count := Anzahl der Elemente ≤x zwischen a[i] und a[j];
order := i+count-1;
Vertausche a[k] mit a[order]; // Jetzt steht x an der richtigen Position
right := j;
for left := 0 to count-2 do
   if a[i+left] > x
      then while a[right] > x do right := right - 1;
           Vertausche a[i+left] mit a[j];
return order;
```

Referenzen zum Nacharbeiten und Vertiefen:
Cormen Kap. 7.1 (deutlich andere Implementierung von Partition)

Algorithmik 1

1.1 Vergleich von grundlegenden Sortiertechniken

Im Detail: Genaue Laufzeitabschätzung von Quicksort: $\Theta(n^2)$

- Untere Laufzeitschranke $\Omega(n^2)$:

  Es wird für jedes n eine Eingabe angegeben, deren Laufzeit in $\Omega(n^2)$ ist

- Obere Laufzeitschranke $O(n^2)$:

  Benutzung der Rekursionsgleichung im Skript
  und Beweis von $T(n) \leq c \bullet n^2$ durch verallgemeinerte vollständige Induktion über n

  Anmerkung:
  Die Behauptung, dass k=1 und k= n die schlechtesten Fälle sind, wird im Skript nicht bewiesen und für den Beweis der Laufzeitschranken auch nicht benötigt.

Referenzen zum Nacharbeiten und Vertiefen:
Alt S. 7,
Cormen Kap. 7.2

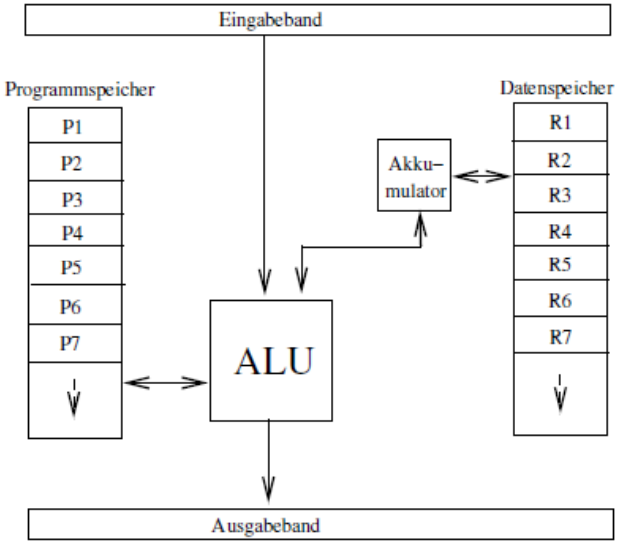
Algorithmik 1

1.2 Einführung in Komplexitätsmaße für Algorithmen

Berechnungsmodell: RAM (Random Access Machine)

- Definition einer RAM

  Kleiner assembler-ähnlicher Befehlssatz,
  Steuerkopf mit Zugriff auf Programmspeicher und Datenspeicher in konstanter Zeit



Befehl	:	auszuführende Operation
LOAD a	:	$R_0 \longleftarrow R_a$
STORE i	:	$R_i \longleftarrow R_0$
ADD a	:	$R_0 \longleftarrow R_0 + R_a$
SUB a	:	$R_0 \longleftarrow R_0 - R_a$
MULT a	:	$R_0 \longleftarrow R_= \cdot R_a$
DIV a	:	$R_0 \longleftarrow \lfloor R_0/R_a \rfloor$
READ i	:	$R_0 \longleftarrow$ aktuelles Inputszeichen
WRITE i	:	Inhalt von $R_i \longrightarrow$ Ausgabeband
JUMP b	:	nächster Befehl ist P_i
JZERO b	:	nächster Befehl ist P_i, wenn $R_0 = 0$
JGZERO b	:	nächster Befehl ist P_i, wenn $R_0 > 0$
HALT	:	Stoppbefehl

aus Skript Lang

Referenzen zum Nacharbeiten und Vertiefen:
Alt S. 11-13
Skript Lang 2007, Kap. 2.6

Algorithmik 1

1.2 Einführung in Komplexitätsmaße für Algorithmen

Berechnungsmodell: RAM (Random Access Machine)

- Kostenmaße

  EKM: Alle Operationen kosten konstant viel Zeit unabhängig von Größe der Operanden.

  LKM: Die Kosten einer Operation hängen von der Größe der Operanden ab.

- Laufzeitäquivalenz

  Algorithmus benötigt auf einer RAM die Zeit $\Theta(f(n))$ (LKM oder EKM)
  $\Leftrightarrow$ Algorithmus benötigt auf einem normalen Computer die Zeit $\Theta(f(n))$.

- Polynomielle Verwandtschaft

  Ein Algorithmus benötigt auf einer RAM die Zeit $\Theta(f(n))$ im LKM
  $\Leftrightarrow$ Algorithmus benötigt auf einer Turingmaschine die Zeit $\Theta(P(f(n)))$ für ein Polynom P.

Referenzen zum Nacharbeiten und Vertiefen:
Alt S. 11-13
Skript Lang 2007, Kap. 2.6

Algorithmik 1

1.2 Einführung in Komplexitätsmaße für Algorithmen

Rechnen mit Landau-Symbolen

- Definition von O, Ω und Θ

  $T(n) \in O(f(n)) \Leftrightarrow \exists c \in \mathbb{R}\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0: T(n) \leq c \cdot f(n)$
  $T(n) \in \Omega(f(n)) \Leftrightarrow \exists c \in \mathbb{R}\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0: T(n) \geq c \cdot f(n)$
  $T(n) \in \Theta(f(n)) \Leftrightarrow \exists c_1, c_2 \in \mathbb{R}\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0: c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$

- Rechenregeln für Landausymbole

  1) $x < y \Rightarrow O(n^x) \subsetneq O(n^y)$

  2) $x > 0 \Rightarrow O(\log n) \subsetneq O(n^x)$

  3) $O(f(n)+g(n)) \in O(f(n)) \cup O(g(n))$ (“Maximum”)

Referenzen zum Nacharbeiten und Vertiefen:
Cormen Kap. 3

Algorithmik 1

1.2 Einführung in Komplexitätsmaße für Algorithmen

Master-Theorem
für die Laufzeitabschätzung von Divide+Conquer-Algorithmen

Sei die Rekursionsgleichung eines Divide+Conquer-Algorithmus gegeben durch:

$$T(n) = a\,T(n/b) + f(n)$$

Dann gilt für $f(n) \in \Theta(n^k)$:

1) $a < b^k \Rightarrow T(n) \in \Theta(n^k)$

2) $a = b^k \Rightarrow T(n) \in \Theta(n^k \log n)$

3) $a > b^k \Rightarrow T(n) \in \Theta(n^{\log_b a})$

Dieselben Resultate gelten für O und Ω

Referenzen zum Nacharbeiten und Vertiefen:
Cormen Kap. 4

Algorithmik 1

1.2 Einführung in Komplexitätsmaße für Algorithmen

Bedeutung der Landausymbole für die Komplexität von Algorithmen

Sei I(A) eine zulässige Eingabe für den Algorithmus A und size(I(A)) die Größe der Eingabe.
Sei $T_A(I(A))$ die Laufzeit (als Operationszähler) des Algorithmus, wenn I(A) die Eingabe ist.

- Obere Laufzeitschranke im schlechtesten Fall:

  A ist ein O(f(n))-Algorithmus $\Leftrightarrow \forall n \in \mathbb{N}\ \forall$ I(A),size(I(A))=n: $T_A(I(A)) \in O(f(n))$
  "Alle Eingaben müssen laufzeitbeschränkt sein."

- Untere Laufzeitschranke im schlechtesten Fall:

  A ist ein Ω(f(n))-Algorithmus $\Leftrightarrow \forall n \in \mathbb{N}\ \exists$ I(A),size(I(A))=n: $T_A(I(A)) \in \Omega(f(n))$
  "Für jedes n gibt es eine Eingabe mit dieser Mindestlaufzeit."

- Exakte asymptotische Laufzeit im schlechtesten Fall:

  A ist ein Θ(f(n))-Algorithmus im schwachen Sinn $\Leftrightarrow$
  A ist ein O(f(n))-Algorithmus und A ist ein Ω(f(n))-Algorithmus

  A ist ein Θ(f(n))-Algorithmus im starken Sinn $\Leftrightarrow \forall n \in \mathbb{N}\ \forall$ I(A),size(I(A))=n: $T_A(I(A)) \in \Theta(f(n))$
  "Alle Eingaben haben genau diese Laufzeit (zwischen 2 Konstanten)."

Referenzen zum Nacharbeiten und Vertiefen: ? (für Hinweise bin ich dankbar)

Algorithmik 1

1.3 Untere Schranken für vergleichsbasierte Algorithmen

- Untere Schranke für das Suchen eines maximalen Elements einer Menge

  Die Menge habe n Elemente (Inputgröße).
  Der Vergleichsgraph muss zusammenhängend sein ➔ mindestens n-1 Vergleiche (Ω(n))
  Einen O(n)-Algorithmus dafür gibt es ➔ Dieser ist optimal.

- Untere Schranke für das Suchen des k-ten Elements einer Menge

  Die Menge habe n Elemente (Inputgröße).
  Der Vergleichsgraph muss zusammenhängend sein ➔ mindestens n-1 Vergleiche (Ω(n))
  Optimaler Algorithmus dafür? ➔ Kapitel 2

- Untere Schranke für das Sortierproblem

  Zusammenhang zwischen Tiefe des Vergleichsbaums und Laufzeit in Vergleichen
  Zusammenhang zwischen Tiefe von binären Suchbäumen und Anzahl der Blätter
  Abschätzung für n!, Folgerung für log (n!) ➔ mindestens Ω(n log n) Vergleiche
  Der Mergesort braucht nur O(n log n) Vergleiche ➔ Dieser ist optimal.

Referenzen zum Nacharbeiten:
Alt S. 17 - 21