Grundlagen der Programmierung

Vorlesung 12
Sebastian Iwanowski
FH Wedel

Entwurf von Algorithmen

Wie klassifiziert man Algorithmen ?

- **offensichtlich nicht durch die Unterscheidung rekursiv / iterativ !**
- **Unterscheidung nach Lösungsstrategie (greedy, divide and conquer, ...) schon besser !**

Welche Implementierungsvariante (rekursiv / iterativ) ist zu einem gegebenen Algorithmus zu bevorzugen ?

Entwurf von Algorithmen

Welche Lösungsstrategie ist für ein gegebenes Problem besser ?

Unterscheiden sich die Algorithmen, die zu einer Lösungsstrategie gehören ?

Wie unterscheidet man zwischen Implementierungsvarianten desselben Algorithmus und zwei verschiedenen Algorithmen ?

Grundlagen der Programmierung

1. Einführung
   - Grundlegende Eigenschaften von Algorithmen und Programmen
2. Logik
   - Aussagenlogik
   - Prädikatenlogik
3. Programmentwicklung und –verifikation
   - Grundlagen der Programmverifikation
   - Zuweisungen und Verbundanweisungen
   - Verzweigungen
   - Schleifen
   - Modularisierung
   - Rekursion
4. Entwurf und Analyse von Algorithmen
   - Klassifikation von Algorithmen
   - Programmierung von Algorithmen
   - ➔ Bewertung von Algorithmen

Bewertung von Algorithmen

Was wird bewertet ?

- benötigte Rechenzeit
- benötigter Speicherplatz

Welche Eigenschaften sollte eine Bewertung haben ?

- unabhängig von der Implementierung
- unabhängig vom eingesetzten Computer

Wie ist das möglich ?

Bewertung von Algorithmen

Der Schlüssel zum Erfolg: Die Turingmaschine

- von Alan Turing 1937 konstruierter theoretischer „Urcomputer“

Für jeden bis heute konstruierten Computer gilt:

- Wenn ein Problem der Größe n
  auf einer Turingmaschine f(n) Rechenschritte braucht,
  dann benötigt es auf einem anderen Computer c·f(n) Rechenschritte.

  Hierbei ist c eine Konstante, die nur vom Computer abhängt und für alle Algorithmen gilt.

Daraus abgeleitetes Grundprinzip:

- **Vernachlässige Konstante,**
  die nicht von der Problemgröße abhängen !

Bewertung von Algorithmen

Definition Komplexitätsklasse:

Ein Algorithmus gehört bzgl. der Rechenzeit (des Speicherplatzes) zur Komplexitätsklasse **O(f(n))**, wenn es eine Konstante c gibt, sodass gilt:

Die Rechenzeit (Der Speicherplatz) für ein Problem der Größe n benötigt maximal c·f(n) Rechenschritte (Speicherplätze).

- c darf nicht von der Problemgröße n abhängen.
- c darf vom Algorithmus abhängen.
- c darf vom Computer und von der Implementierung abhängen.
- O wird das Landau-Symbol genannt (nach Edmund Landau, 1877-1938)

Typische Komplexitätsklassen von Algorithmen:

O(1), O(log n), O(n), O(n log n), O (n^2), O(2^n)

O(P(n)), wobei P ein Polynom ist

Bewertung von Algorithmen

Bewertung von Algorithmen

Gegeben ein Algorithmus:
Finde die günstigste Komplexitätsklasse bzgl. **Laufzeit** und Speicherplatz:

- **im ungünstigsten Fall (worst case)**
- im Durchschnittsfall (average case)

Beispiele:

	Laufzeit:	Speicherplatz:
Lineare Suche:	O(n)	O(n)
„DC-Suche“:	O(n)	O(n)
Binärsuche:	O(log n)	O(n)
Selectionsort:	$O(n^2)$	O(n)
Mergesort:	O(n log n)	O(n)
Quicksort:	$O(n^2)$ **w.c.** O(n log n) **a.c.**	O(n)

Bewertung von Algorithmen

Bewertung von Problemen

Gegeben ein Problem:
Finde die günstigste Komplexitätsklasse, zu der es einen Algorithmus gibt, der das Problem ***im allgemeinen Fall*** löst.

Ein Problem gehört bzgl. der Rechenzeit (des Speicherplatzes) zur Komplexitätsklasse **Ω(f(n))**, wenn es eine Konstante c gibt, so dass gilt:

Die Rechenzeit (Der Speicherplatz) ***jedes*** Algorithmus für das Problem der Größe n benötigt ***mindestens*** c·f(n) Rechenschritte (Speicherplätze).

Beispiele:

	Laufzeit:	Speicherplatz:
Allgemeine Suche:	Ω(n)	Ω(n)
Allgemeines Sortieren:	Ω(n log n)	Ω(n)

NP-Vollständigkeit

NP-vollständige Probleme sind folgendermaßen charakterisiert:

- Das Problem ist auf einer (hypothetischen) **nichtdeterministischen** Turingmaschine in polynomialer Zeit (O(P(n) für ein Polynom P) lösbar.

- Jedes NP-vollständige Problem ist **genau dann**
  auf einer normalen Turingmaschine in polynomialer Zeit lösbar,
  wenn *alle* NP-vollständigen Probleme
  auf einer normalen Turingmaschine in polynomialer Zeit lösbar sind.

Offene Frage der Informatik:

1) Sind NP-vollständige Probleme auf einer normalen Turingmaschine in polynomialer Zeit lösbar ?

2) Gehören sie zu einer Komplexitätsklasse Ω(f(n)),
   wobei f(n) stärker wächst als jedes Polynom P(n) ?

Was ist für die Klausur relevant ?

Vorlesung 1: Überblick / Grundlegende Eigenschaften

Grundlegende Begriffe und Konzepte: Funktion, Spezifikation, Algorithmus, Programm, Verifikation, Konstruktion und alles, was damit zusammenhängt. Fähigkeit, die Konzepte an Minibeispielen anzuwenden.

Vorlesung 2: Aussagenlogik

Grundlegende Begriffe wie Aussage, Wahrheitswert, Formel, Belegung, Erfüllbarkeit. Sicherer Umgang mit aussagenlogischen Umformungen, eindeutige und exakte Notation. Konjunktive Normalform: Begriffe und Fähigkeit der Umformung in eine solche.

Vorlesung 3-4: Prädikatenlogik

Unterschied zur Aussagenlogik: Variable, Prädikate, Funktionen, Quantoren. Erklären dieser Begriffe an Beispielen, aktive und passive Beherrschung der Notation. Rechenregeln für Quantoren, Anwendung der Prädikatenlogik auf Minibeispiele aus dem täglichen Leben und der Arithmetik.

Was ist für die Klausur relevant ?

Vorlesung 5: Grundlagen der Programmverifikation

Hoare-Tripel: Aktive und passive Beherrschung der Notation. Zusammenhang zwischen unterschiedlichen Stärken von Vorbedingungen und Nachbedingungen, Exakte Verifikation kleinster Programmteile: Finden von stärksten Nachbedingungen und schwächsten Vorbedingungen.

Vorlesung 6: Verzweigungen

Finden von stärksten Nachbedingungen und schwächsten Vorbedingungen in Anwendungsbeispielen.

Vorlesung 7-8: Schleifen

Bestimmung und Verifikation von Invariantenbedingungen sowie von Variantenzahlen für die Terminierung, Verifikation von kleinen Schleifen entweder mit vollständiger Induktion oder mit dem Invarianten/Variantenverfahren

Vorlesung 8: Modularisierung

Parameter, Rückgabewerte: Erklärung der Bedeutung und Verwendung, Anwendung in Minibeispielen. Call-by-reference und call-by-value: Erklärung der Unterschiede, Vorteile und Nachteile der beiden Parameterübergabetechniken.

Was ist für die Klausur relevant ?

Vorlesung 9-10: Rekursion

Nachvollziehen des Programmablaufs vorgegebener rekursiver Programme, Bestimmen von notwendigen Vorbedingungen.
Erklärung der Konzepte primitiv rekursiv, endrekursiv und linear rekursiv. Erkennen dieser Konzepte an Beispielen. Umformung von endrekursiven Programmen in Schleifen. Vor- und Nachteile der rekursiven Formulierung gegenüber der iterativen.

Vorlesung 11-12: Entwurf und Analyse von Algorithmen

Grundideen der Algorithmen Lineare Suche, Binärsuche, Selectionsort und Mergesort, Kenntnis der Komplexitätseigenschaften dieser Algorithmen, Begründung, warum Komplexitätsklassen ein gutes Maß zur Bewertung von Algorithmen sind (zum Beispiel im Vergleich zu iterativ / rekursiv).

Das war die Vorlesung
Grundlagen der Programmierung