technische universität
dortmund

LS 2 /
Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Organisatorisches

Vorlesung DAP2

- Dienstag 12-14 c.t.
- Donnerstag 14-16 c.t.

Zu meiner Person

- Christian Sohler
- Fachgebiet: Komplexitätstheorie und effiziente Algorithmen
- Lehrstuhl 2, Informatik

Organisatorisches

Übungen

- Montag: 14-16, 16-18
- Dienstag: 10-12, 14-16, 16-18
- Mittwoch: 12-14, 14-16, 16-18
- Donnerstag: 12-14, 16-18

- Zum Teil mehrere parallele Gruppen
- Anmeldung über AsSESS
- Beginn der Anmeldung: Dienstag im Laufe des Tages
- Anmeldeschluss: Donnerstag, 18 Uhr
- Änderungen der Übungsgruppe: Bis Montag 10 Uhr

Organisatorisches

Übungen

- Präsenzübungen/ Heimübungen abwechselnd
- Abgabe Heimübung: Freitag 10 Uhr Briefkästen im Pav.6
- Zulassung zur Klausur (Studienleistung Übung; Teil 1):
  - Teilnahme an den Übungen
  - 50% der Übungspunkte
- Max. 3 Personen pro Übungsblatt

Organisatorisches

Übungen Praktikum

- Für Studierende des Bachelorstudiengangs verpflichtend
- Bachelor Elektrotechnik/Informationstechnik und Informations- und Kommunikationstechnik hat eigenes Praktikum
- Termine:
- Mittwoch 12-14, 14-16, 16-18
- Donnerstag 8-10, 10-12, 12-14
- Freitag 8-10, 10-12

- Anmeldung über AsSESS (ab Dienstag; Anmeldeschluss Do. 18 Uhr; Änderungen bis Montag 10 Uhr)

Organisatorisches

Übungen Praktikum

- Heimübungen, Präsenzübungen
- Zulassung zur Klausur (Studienleistung Praktikum):
- 8 von insgesamt 14 Punkten bei den 7 Präsenzaufgaben
- 7 von insgesamt 12 Punkten bei den 6 Heimaufgaben
- Anwesenheit bei den Übungen

Sonstiges

- Schüler -> Tobias Marschall
- Poolräume können außerhalb der Veranstaltungszeiten immer genutzt werden
- Weitere Informationen auf der Webseite http://ls11-www.cs.uni-dortmund.de/teaching/dap2praktikum

Organisatorisches

Tests

- 1. Test: 18. Mai
- 2. Test: mal schauen
- Einer der beiden Test muss mit 50% der Punkte bestanden werden (Studienleistung Übung; Teil 2)

Organisatorisches

Bei Fragen

- Meine Sprechzeiten: Mo 14-15 Uhr oder nach der Vorlesung
- Organisatorische Fragen an
- Vorlesung:
- Melanie Schmidt (melanie.schmidt@tu-dortmund.de)
- Christiane Lammersen (christiane.lammersen@tu-dortmund.de)

- Praktikum:
- Tobias Marshall (tobias.marschall@tu-dortmund.de)

- Außerdem INPUD Forum

Organisatorisches

Klausurtermine

- 10.8. 17-21 Uhr
- 21.9. 17-21 Uhr

Weitere Infos

- Vorlesungsseite
  http://ls2-www.cs.tu-dortmund.de/lehre/sommer2010/dap2/
- Oder von der Startseite des LS 2 -> Teaching -> DAP2

Einige Hinweise/Regeln

Klausur

- Eine Korrelation mit den Übungsaufgaben ist zu erwarten

Laptops

- Sind in der Vorlesung nicht zugelassen

Literatur

Skripte

- Kein Vorlesungsskript
- Skripte der vergangenen Jahre

Bücher

- Cormen, Leisserson, Rivest: Introduction to Algorithms, MIT Press
- Kleinberg, Tardos: Algorithm Design, Addison Wesley

WWW

- Kurs „Introduction to Algorithms“ am MIT. Online Material (Folien, Video und Audio Files!)
- http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/

Was ist ein Algorithmus?

Informale Definition

- Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung von Instanzen eines Problems in endlich vielen Schritten.

Bemerkung

- Es gibt viele Beschreibungen desselben Algorithmus
- Algorithmus ist unabhängig von der Programmiersprache!!!
- Die Definition wird im nächsten Semester formalisiert

Was ist ein Algorithmus?

Beispiel

- Problem: Maximumsuche
- Instanz: Menge $A=\{a_1,\ldots,a_n\}$ von n Zahlen (als Feld A gegeben)
- Ausgabe: Index i der größten Zahl in A

Algorithmus-Max-Search(Array A)

1. max $\leftarrow$ 1
2. **for** j $\leftarrow$ 2 **to** length(A) **do**
3.   **if** A[j] > A[max] **then** max $\leftarrow$ j
4. **return** max

Was ist eine Datenstruktur?

Informale Definition

- Eine Datenstruktur ist eine Anordnung von Daten, die die Ausführung von Operationen (z.B. Suchen, Einfügen, Löschen) unterstützt.

Einfache Beispiele

- Feld
- sortiertes Feld
- Liste

Lernziele

- *Bewertung von Algorithmen und Datenstrukturen*
  - Laufzeitanalyse
  - Speicherbedarf
  - Korrektheitsbeweise
- *Kenntnis grundlegender Algorithmen und Datenstrukturen*
  - Sortieren
  - Wörterbücher
  - Graphalgorithmen
- *Kenntnis grundlegender Entwurfsmethoden*
  - Teile und Herrsche
  - gierige Algorithmen
  - dynamische Programmierung

Motivation

Beispiele für algorithmische Probleme

- Internetsuchmaschinen
- Berechnung von Bahnverbindungen
- Optimierung von Unternehmensabläufen
- Datenkompression
- Computer Spiele
- Datenanalyse

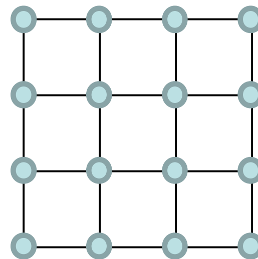
Alle diese Bereiche sind (immer noch) Stoff aktueller Forschung im Bereich Datenstrukturen und Algorithmen

Motivation

Problembeschreibung

- Ein m x n- Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17x17 Gitter (289$ Problem)

- Beispiel:
(4x4 Gitter)



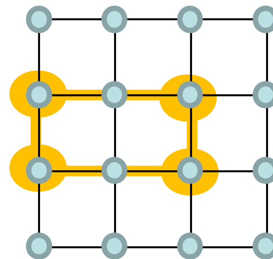
- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

Motivation

Problembeschreibung

- Ein m x n- Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17x17 Gitter (289$ Problem)

- Beispiel:
  (4x4 Gitter)
- Die vier unterlegten Knoten dürfen z.B. nicht alle dieselbe Farbe haben



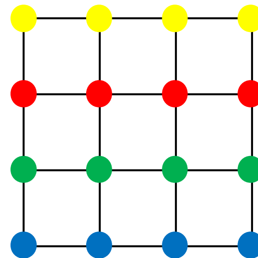
- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

Motivation

Problembeschreibung

- Ein m x n- Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17x17 Gitter (289$ Problem)

- Beispiel:
(4x4 Gitter)



4x4 Gitter ist 4-färbbar!
Geht es besser?

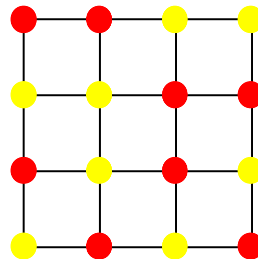
- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

Motivation

Problembeschreibung

- Ein m x n- Gitter heißt c-färbbar, wenn man seine Knoten mit c Farben so färben kann, dass kein am Gitter orientiertes achsenparalleles Rechteck alle Eckknoten in derselben Farbe hat
- Aufgabe: Finde eine 4-Färbung für ein 17x17 Gitter (289$ Problem)

- Beispiel:
(4x4 Gitter)



Ja! 4x4 Gitter ist 2-färbbar!

- http://blog.computationalcomplexity.org/2009/11/17x17-challenge-worth-28900-this-is-not.html

Motivation

17x17 Problem

- Es ist z.Z. nicht möglich, das 17x17 Problem mit einem Rechner zu lösen
- Warum ist dieses Problem so schwer zu lösen?
- Es gibt sehr viele Färbungen!

Fragen/Aufgaben

- Können wir die Laufzeit eines Algorithmus vorhersagen?
- Können wir bessere Algorithmen finden?

Algorithmenentwurf

Anforderungen

- Korrektheit
- Effizienz (Laufzeit, Speicherplatz)

Entwurf umfasst

1. Beschreibung des Algorithmus/der Datenstruktur
2. Korrektheitsbeweis
3. Analyse von Laufzeit und Speicherplatz

Algorithmenentwurf

Warum mathematische Korrektheitsbeweise?

- Fehler können fatale Auswirkungen haben (Steuerungssoftware in Flugzeugen, Autos, AKWs)
- Fehler können selten auftreten („Austesten“ funktioniert nicht)

Der teuerste algorithmische Fehler?

- Pentium bug (> $400 Mio.)
- Enormer Image Schaden
- Trat relativ selten auf

Algorithmenentwurf

Warum Laufzeit/Speicherplatz optimieren?

- Riesige Datenmengen durch Vernetzung (Internet)
- Datenmengen wachsen schneller als Rechenleistung und Speicher
- Physikalische Grenzen
- Schlechte Algorithmen versagen häufig bereits bei kleinen und mittleren Eingabegrößen

1. Teil der Vorlesung – Grundlagen der Algorithmenanalyse

Inhalt

- Wie beschreibt man einen Algorithmus?
- Wie beweist man die Korrektheit eines Algorithmus?
- Rechenmodell
- Laufzeitanalyse

Pseudocode

- Beschreibungssprache ähnlich wie C, Java, Pascal, etc…
- Hauptunterschied: Wir benutzen immer die klarste und präziseste Beschreibung
- Manchmal kann auch ein vollständiger Satz die beste Beschreibung sein
- Wir ignorieren Software Engineering Aspekte wie
  - Modularität
  - Fehlerbehandlung

Sortieren

- Problem: Sortieren
- Eingabe: Folge von n Zahlen $(a_1,\ldots,a_n)$
- Ausgabe: Permutation $(a'_1,\ldots,a'_n)$ von $(a_1,\ldots,a_n)$, so dass $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Beispiel:

- Eingabe: 15, 7, 3, 18, 8, 4
- Ausgabe: 3, 4, 7, 8, 15, 18

Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

Beschreibung des Algorithmus in Pseudocode (kein C, Java, etc.)

Insertion Sort

Pseudocode

- Schleifen (**for**, **while**, **repeat**)
- Zuweisungen durch ←
- Variablen (z.B. i, j, key) sind lokal definiert
- Keine Typdeklaration, wenn Typ klar aus dem Kontext
- Zugriff auf Feldelemente mit [.]
- Verbunddaten sind typischerweise als Objekte organisiert
- Ein Objekt besteht aus Attributen oder Ausprägungen
- Beispiel: Feld wird als Objekt mit Attribut Länge betrachtet
- Beispiel: Objekt ist Graph G mit Knotenmenge V
- Auf die Ausprägung V von Graph G wird mit V[G] zugegriffen

Insertion Sort

Pseudocode

- Objekte werden als Zeiger referenziert, d.h. für alle Ausprägungen f eines Objektes x bewirkt y ← x das gilt: f[y] = f[x].
- Blockstruktur durch Einrücken
- Bedingte Verzweigungen (**if then else**)
- Prozeduren „call-by-value“ ; jede aufgerufene Prozedur erhält neue Kopie der übergebenen Variable
- Die lokalen Änderungen sind nicht global sichtbar
- Bei Objekten wird nur der Zeiger kopiert (lokale Änderungen am Objekt global sichtbar)
- Rückgabe von Parametern durch **return**
- Kommentare durch ➢

Insertion Sort

InsertionSort(Array A)
1. **for** j ← 2 **to** length[A] **do**
2. key ← A[j]
3. i ← j-1
4. **while** i>0 and A[i]>key **do**
5. A[i+1] ← A[i]
6. i ← i-1
7. A[i+1] ← key

- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke

Beispiel (siehe Vollversion)

8	15	3	14	7	6	18	19

InsertionSort

- Wir haben beobachtet, dass Algorithmus InsertionSort korrekt sortiert
- Wie können wir zeigen, dass dies für jede Eingabe stimmt?
  -> Korrektheitsbeweise

Zusammenfassung

- Es gibt schwierige algorithmische Probleme, die bzgl. Laufzeit und Speicherbedarf optimierte Algorithmen benötigen
- Korrektheitsanalyse von Algorithmen ist notwendig
- Wir beschreiben Algorithmen in Pseudocode