Algorithmik

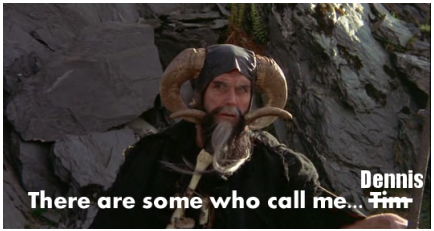
Vorkurs Informatik WS 2018/2019

Dennis Aumiller
Aumiller@stud.uni-heidelberg.de

10.10.2018

Über mich

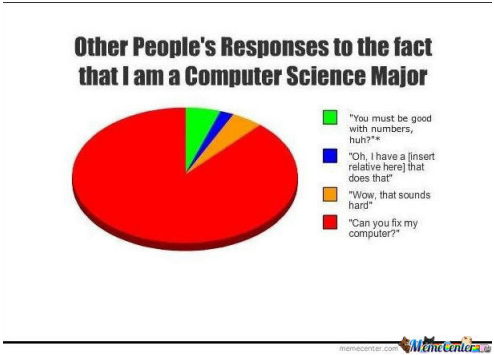
- 11. Semester, gerade an meiner Masterarbeit
- Mein Büro befindet sich direkt neben der Fachschaft (Mathematikon, 1/304)!
- jederzeit für Fragen erreichbar: Aumiller@stud.uni-heidelberg.de



Über den Vorkurs

Viele Menschen haben ein falsches Bild von der Informatik.
Informatik ist nicht:

- Computer reparieren
- (nur) programmieren
- Webseiten hacken



Definition Informatik:

"Die Wissenschaft, die sich mit Computern und ihrer Anwendung (im Rahmen der elektronischen Datenverarbeitung) beschäftigt." - Wikipedia

Im Englischen ist der Unterschied klarer:
IT (Information Technology) vs Computer Science

3 goldene Regeln für das (Informatik-)Studium

- **Fragt nach!!!!!!**
- Wenn ein Professor fragt “Haben Sie das verstanden?”, haben es die meisten *nicht* verstanden
- Geht in die Vorlesungen
- Bildet Lerngruppen

Jetzt dürft ihr euch kurz “vorstellen...”

- Wer hatte schon in der Schule Informatik?
- Habt ihr dort schon programmiert? (mit welchen Sprachen?)
- Wieso studiert ihr Informatik?

Schritt für Schritt zum Ziel

Der beste Vergleich ist ein Kochrezept:

- beschreibt in welcher Reihenfolge
- mit welchen Zutaten
- was passiert
- in klarer Abfolge
- endlich viele Schritte

“but wait, there’s more”

Ein Kochrezept ist sehr beschränkt:

- Was passiert bei anderen Zutaten?
- Wie allgemein kann ein Rezept sein?

Algorithmen in der Informatik

Simpelste Algorithmen?

Algorithmen in der Informatik

Simpelste Algorithmen?

- Addition, Subtraktion, Multiplikation, Division

Algorithmen in der Informatik

Simpelste Algorithmen?

- Addition, Subtraktion, Multiplikation, Division
- Inkrement

Algorithmen in der Informatik

Simpelste Algorithmen?

- Addition, Subtraktion, Multiplikation, Division
- Inkrement
- Lösen eines Gleichungssystems

Algorithmen in der Informatik

Simpelste Algorithmen?

- Addition, Subtraktion, Multiplikation, Division
- Inkrement
- Lösen eines Gleichungssystems
- Near-optimal isosurface extraction using span-space

Am Ende des Tages bleibt die Grundstruktur gleich

Es gibt allerdings Ausnahmen:

Algorithmen in der Informatik

Simpelste Algorithmen?

- Addition, Subtraktion, Multiplikation, Division
- Inkrement
- Lösen eines Gleichungssystems
- Near-optimal isosurface extraction using span-space

Am Ende des Tages bleibt die Grundstruktur gleich

Es gibt allerdings Ausnahmen:

- Randomisierte Algorithmen (Info Master, Prof. Merkle)
- Neuromorphic Computing (Physik, Prof. Meier)
- Machine Learning / Neural Networks (Info Master, div. Veranstaltungen)

Folgerung

- Wie wir sehen ist der Computer “beschränkt” in seinen Möglichkeiten.

Folgerung

- Wie wir sehen ist der Computer “beschränkt” in seinen Möglichkeiten.
- Daher ist es Aufgabe eines Informatikers, ihm die Schritte vorherzugeben.

Grundlegende Programmstrukturen

- if-Statements

Grundlegende Programmstrukturen

- if-Statements
- for-Schleifen

Grundlegende Programmstrukturen

- if-Statements
- for-Schleifen
- while-Loops

Grundlegende Programmstrukturen

- if-Statements
- for-Schleifen
- while-Loops
- sogenannte *atomare* Operationen

Pseudocode

- Oftmals werden Algorithmen oder Konzepte in der Form von Pseudocode vorgestellt
- Hilfreicher und übersichtlicher, da unterschiedliche Programmiersprachen
- das Konzept ist ja immernoch dasselbe

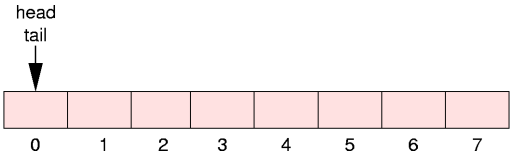
einfaches Beispiel

Data: array N
Result: sorted array N
initialization;
while N *is not sorted* **do**
    read current;
    if *some condition* **then**
        describe steps;
    else
        different steps;
    end
end

Algorithm 1: Beispiel eines Pseudocode-Algorithmus

Arrays/Listen

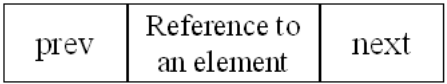
Ein Array ist eine (meist in Reihe abgespeicherte) Ansammlung gleicher Datentypen, identifiziert bei einem eindeutigen Schlüssel.

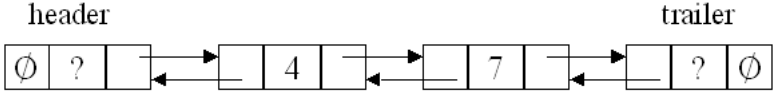


- Ideal zum Speichern bei vielen gleichen Daten
- Temperaturdaten, Sensordaten, ...

Arrays/Listen

- Listen haben zusätzlich Informationen über den Nachfolger und/oder Vorgänger gespeichert.
- Dies ermöglicht eine höhere Flexibilität bei der Speicherzuweisung und dem Löschen oder Hinzufügen von Daten





Objektorientierte Programmierung: Klassen

Um Objekte mit vielerlei Eigenschaften zu speichern und diese nach Bedarf anzulegen, gibt es Klassen.
Versucht euch ein Auto möglichst abstrakt vorzustellen:

- Räder

Objektorientierte Programmierung: Klassen

Um Objekte mit vielerlei Eigenschaften zu speichern und diese nach Bedarf anzulegen, gibt es Klassen.
Versucht euch ein Auto möglichst abstrakt vorzustellen:

- Räder
- Chassis

Objektorientierte Programmierung: Klassen

Um Objekte mit vielerlei Eigenschaften zu speichern und diese nach Bedarf anzulegen, gibt es Klassen.
Versucht euch ein Auto möglichst abstrakt vorzustellen:

- Räder
- Chassis
- Motor

Objektorientierte Programmierung: Klassen

Um Objekte mit vielerlei Eigenschaften zu speichern und diese nach Bedarf anzulegen, gibt es Klassen.
Versucht euch ein Auto möglichst abstrakt vorzustellen:

- Räder
- Chassis
- Motor
- ...

Divide and Conquer

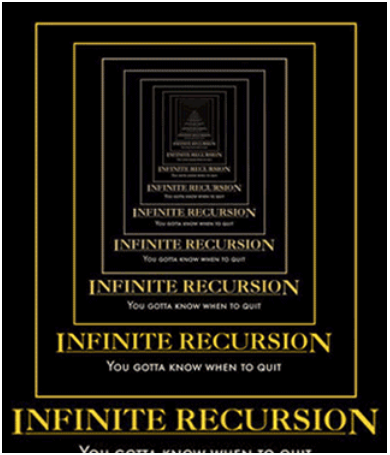
Divide and Conquer beschreibt eine häufig angewandte Strategie in der Informatik: Anstatt ein sehr komplexes Problem "direkt" zu lösen wird es unterteilt, bis es aus sehr vielen kleinen, aber einfach zu lösenden Problemen besteht.
Die Vorteile liegen auf der Hand:

- Heutzutage hoher grad an Parallelität wünschenswert, um Kapazitäten auszunutzen
- Unterteilung oft sehr leicht

Rekursion

Ähnlich wie Divide and Conquer wird das Problem so lange auf sich selbst zurückgeführt, bis es sehr leicht zu lösen ist:

```
function recursive(int n) {
    if (n == 1) {
       return 1;
    }
    return n * recursive(n-1);
}
```



Komplexität von Algorithmen: The "How and When"

Um die Qualität eines Algorithmus zu bemessen, misst man die **Laufzeit** oder **Komplexität** eines Algorithmus

- Die Laufzeit von Algorithmen hängt stark von guter Programmierung ab
- Unnötige Schritte mögen bei "normalen" Testfällen unauffällig wirken
- Bei sehr großen Eingabedaten steigt die Laufzeit dann aber drastisch an!

Bestimmung der Komplexität anhand der Max-Funktion 1

```
for i = 1, ..., n do
    x = L_i;
    isMax = true;
    for j = 1, ..., n do
        y = L_j ;
        if a < b then
            isMax = false;
        end
    end
    if isMax then
        max = a;
    end
end
```

Algorithmus 1

input data: unsorted array L
output data: maximum of L
n = |L|

Bestimmung der Komplexität anhand der Max-Funktion 2

```
for i = 1, ..., n do
    x = L_i;
    isMax = true;
    for j = 1, ..., n do
        y = L_j ;
        if a < b then
            isMax = false;
        end
    end
    if isMax then
        max = a;
        return max;
    end
```

Algorithmus 2

input data: unsorted array L
output data: maximum of L
n = |L|

Bestimmung der Komplexität anhand der Max-Funktion 3

```
for i = 1, ..., n do
    x = L_i;
    isMax = true;
    for j = i + 1, ..., n do
        y = L_j ;
        if a < b then
            isMax = false;
        end
    end
    if isMax then
        max = a;
        return max;
    end
```

Algorithmus 3

input data: unsorted array L
output data: maximum of L
n = |L|

Bestimmung der Komplexität anhand der Max-Funktion 4

$max = L_1$;
for $i = 2, ..., n$ **do**
  if $max < L_i$ **then**
    $max = L_i$;
  end
end
return max;

Algorithmus 4

input data: unsorted array L
output data: maximum of L
n = |L|

Laufzeitenvergleich

Eingabe	Algo1	Algo2	Algo3	Algo4
$[1, ..., 10]$	$3 * 10^{-6}s$	$3 * 10^{-6}s$	$7 * 10^{-7}s$	$2 * 10^{-7}s$
$[1, ..., 10^2]$	$7 * 10^{-5}s$	$3 * 10^{-6}s$	$1 * 10^{-5}s$	$4 * 10^{-7}s$
$[1, ..., 10^4]$	**0.5 s**	**0.5 s**	$1 * 10^{-3}s$	$3 * 10^{-5}s$
$[1, ..., 10^8]$	**57 d**	**0.5 s**	**28 d**	$0.2s$

Das Ergebnis des Algorithmus ist in allen 4 Fällen natürlich das gleiche! Aber es zeigen sich große Diskrepanzen bei der Laufzeit.

Das Einsetzen beginnt...

Um die Komplexität zu bestimmen, wird nun die Anzahl atomarer Operationen in Abhängigkeit von N bestimmt.

Schematische Beispiel an Algo1

```
for i = 1, ..., n do
    1
    1
    for j = 1, ..., n do
        1
        if a < b then
            1
        end
    end
    if isMax then
        1
    end
end
1
```

Algorithmus 1

input data: unsorted array L
output data: maximum of L
n = |L|

Schematische Beispiel an Algo1

for $i = 1, ..., n$ **do**
  1
  1
  for $j = 1, ..., n$ **do**
    1
    2
  end
  2
end
1

Algorithmus 1

input data: unsorted array L
output data: maximum of L
$n = |L|$

Schematische Beispiel an Algo1

for $i = 1, ..., n$ **do**
| 1
| 1
| 3n
| 2
| $(= 3n + 4)$
end
1

Algorithmus 1

input data: unsorted array L
output data: maximum of L
n = |L|

Die Laufzeiten im Vergleich:

Algo1:
$3n^2 + 4n + 1$

Algo2:
$3n^2 + 4n + 1$
Hierbei handelt es sich allerdings nur um den Worst-Case. Was das bedeutet, kommt gleich noch.

Algo3:
$3/2n^2 + 3/2n + 1$

Algo4:
$2n + 1$

Gedanken zu den Beobachtungen

- Überlegt euch, wie stark die Anzahl an Operationen für ein weiteres Element zunimmt
- Gibt es bestimmte Anordnungen, für die einer der Algorithmen besser funktioniert?
- Hierbei unterscheidet man im wesentlichen zwischen best-case-Verhalten und worst-case-Verhalten. Oftmals hilft es sich zu vergewissern, wie die Daten tatsächlich in der "realen Welt" dann vorliegen, um ein Verhalten zu beurteilen.

Die Landau-Notation

Da die bisherigen Darstellungen sehr stark der Grenzwertanalyse aus der Schule ähneln, wollen wir die Komplexität noch etwas weiter vereinfachen und in "Klassen" einordnen. Hierfür würde die Landau-Notation, insbesondere das $\mathcal{O}$ eingeführt.
Die allgemeine Definition hierfür lautet:
Für $f : \mathbb{N} \to \mathbb{N}$ eine natürliche Abbildung gilt:

$$\mathcal{O}(f) = \{g : \mathbb{N} \mid \exists m \, \exists c \, \forall n : g(n) \leq m \cdot f(n) + c\}$$

Man sagt, dass $\mathcal{O}(f)$ "in etwa so schnell wächst wie g".

Die Landau-Notation

Neben $\mathcal{O}$ gibt es noch weitere Symbole, die für diverse andere Abschätzungen stehen. Die beiden vermutlich wichtigsten sind hierbei noch:

- o: Für $g \in o(f)$, g wächst *langsamer* als f.
- Θ: Für $g \in \Theta(f)$, g wächst *genauso schnell* wie f.

Gedanken zu den Beobachtungen

- Überlegt euch, wie stark die Anzahl an Operationen für ein weiteres Element zunimmt
- Gibt es bestimmte Anordnungen, für die einer der Algorithmen besser funktioniert?
- Hierbei unterscheidet man im wesentlichen zwischen best-case-Verhalten und worst-case-Verhalten. Oftmals hilft es sich zu vergewissern, wie die Daten tatsächlich in der "realen Welt" dann vorliegen, um ein Verhalten zu beurteilen.

Die Landau-Notation am Beispiel

Algo1 $\in \mathcal{O}(n^2)$

Algo2 $\in \mathcal{O}(n^2)$

Algo3 $\in \mathcal{O}(n^2)$

Algo4 $\in \mathcal{O}(n)$

Weitere häufig auftretende Klassen sind $\in \mathcal{O}(n \cdot log\ n)$, $\in \mathcal{O}(log\ n)$

Wenn euer Algorithmus auch nur Richtung $\in \mathcal{O}(n^3)$ geht, macht ihr (meistens) was falsch! Eine Ausnahme bilden hier NP-vollständige Probleme.

Sortieralgorithmen

Wie würdet ihr schematisch ein Array von Zahlen sortieren?
Schlagwörter für Interessierte: BubbleSort, MergeSort, Quicksort

Quellen

Inhalte der Vorlesung stellen sich aus folgenden Quellen zusammen:

- Vorkurs Informatik WS2015/2016, credit Lutz Büch
- Wikipedia bzgl. Definitionen
- Vorlesung “Einführung in die praktische Informatik” (IPI)
- Vorlesung “Algorithmen und Datenstrukturen” (IAD)
- Mathevorlesung “Lineare Algebra 1” (LA1)