Algorithmen und Programmierung

Kapitel 6
Eigenschaften von Algorithmen

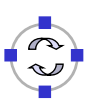
Überblick

Berechenbarkeit und Entscheidbarkeit

Korrektheit von Algorithmen

Komplexität

Anhang – mathematische Hilfsmittel:
- Vollständige Induktion
- Rekurrenzen



Berechenbarkeit und Entscheidbarkeit

- Frage: Gibt es Problemstellungen, für die kein Algorithmus zur Lösung existiert?
- Oder: Kann man alles berechnen / programmieren?
- Berechenbarkeit:

> Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt berechenbar, wenn es einen Algorithmus gibt, der für Eingaben $(x_1, \ldots, x_k) \in \mathbb{N}^k$ terminiert, d. h. in endlicher Zeit $f(x_1, \ldots, x_k)$ berechnet.

- Menge dieser Funktionen entspricht allen jemals mit Computern berechenbaren Funktionen (→ CHURCH'sche These)

Existenz nichtberechenbarer Funktionen

- Nicht jede Funktion ist berechenbar
- Folgt aus Gödelschen Unvollständigkeitssatz:

> Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig.

- Beweisidee:
  - Abzählung der Sätze des formalen Systems
  - Aussage konstruieren: „Satz x ist nicht beweisbar“
  - entweder: Satz x ist wahr, dann nicht beweisbar
  - oder: Satz x ist falsch, dann beweisbar und demnach wahr
    ⇝ Widerspruch
- Kann auch aus der Unlösbarkeit des sogenannten *Halteproblems* (siehe unten) gefolgert werden

Beweisskizze: Existenz nichtberechenbarer Funktionen I

- Eigenschaft von Algorithmen:

> *Jeder Algorithmus lässt sich durch einen endlichen Text über einem festen, endlichen Alphabet beschreiben.*

- Texte über Alphabet A = $\{a_1, a_2, ..., a_n\}$:

$$A^* = \{ \epsilon, a_1, \ldots, a_n, a_1a_1, a_1a_2, \ldots, a_5a_3a_3a_1a_5a_2, \ldots \}$$

- Aufzählung der Wortlänge nach, bei gleicher Länge lexikographisch
  $\Rightarrow$ Menge der Zeichenketten abzählbar
  (d.h. bijektiv auf natürliche Zahlen abbildbar)

Beweisskizze: Existenz nichtberechenbarer Funktionen II

- Cantor'sches Diagonalverfahren
- Menge der Algorithmentexte ist abzählbar (Zeichenketten)
- Beweisbar: Menge der Funktionen ist überabzählbar
- Diagonalbeweis:
  - Betrachte speziell einstellige Funktionen F auf $\mathbb{Z}$, $f : \mathbb{Z} \rightarrow \mathbb{Z}$
  - Annahme: F sei abzählbar, also: $F = \{ f_0, f_1, \ldots \}$
  - Ist folgendes g in F enthalten?

$$g(x) = f_{abs(x)}(x) + 1$$

- Widerspruch!

Konkrete nichtberechenbare Funktionen I

- Vorbemerkungen
  1. Berechnet werden partielle Funktionen $f : A^* \rightarrow A^*$ über einem festen Alphabet A.
  2. Auch die Algorithmen selbst lassen sich als Text über A darstellen.
- Alternative: „Gödelisierung“
  - Kodierung von Algorithmentexten als Zahlen

Konkrete nichtberechenbare Funktionen II

- $x \in A^*$: ein beliebiger Text.
- φ_x : die *vom Algorithmus mit Text x berechnete Funktion*.
  - x kein sinnvoller Algorithmentext → φ_x überall undefiniert.
- Sei $f : A^* \rightarrow A^*$ eine partielle Funktion. Dann ist

$$dom\ f := \{ x \in A^* \mid f(x) \text{ ist definiert} \}$$

der *Urbildbereich* von *f* (*dom* für „domain“).

Die (totale) Funktion $h : A^* \rightarrow A^*$,

$$h(x) = \begin{cases} \epsilon, & \text{falls } x \in dom\ \varphi_x \\ a & \text{sonst (mit a} \in \text{A)} \end{cases}$$

ist nicht berechenbar.

Halteproblem

- Funktion *h* basiert auf einem unentscheidbaren Problem

> Kann man ein Programm entwickeln, das als Eingabe den Quelltext eines zweiten Programms sowie dessen Eingabewerte erhält und entscheiden kann, ob dieses zweite Programm terminiert?

Spezielles Halteproblem

- *h* entscheidet durch die Ausgaben ϵ oder a, ob $x \in dom\ \varphi_x$ ist oder nicht.
- $y \in dom\ \varphi_x$ bedeutet, dass $\varphi_x(y)$ definiert ist, und dies wiederum heißt, dass der Algorithmus mit Text *x* bei Eingabe von *y* irgendwann anhält.
- $x = y$: Anwendung eines Algorithmus auf die *eigene* Beschreibung.
- In diesem Fall drückt die Funktion *h* die folgende Frage aus:

> *„Hält ein Algorithmus irgendwann an, wenn man ihm seinen eigenen Text als Eingabe übergibt?“*
>
> Diese Frage ist auch als spezielles Halteproblem bekannt.

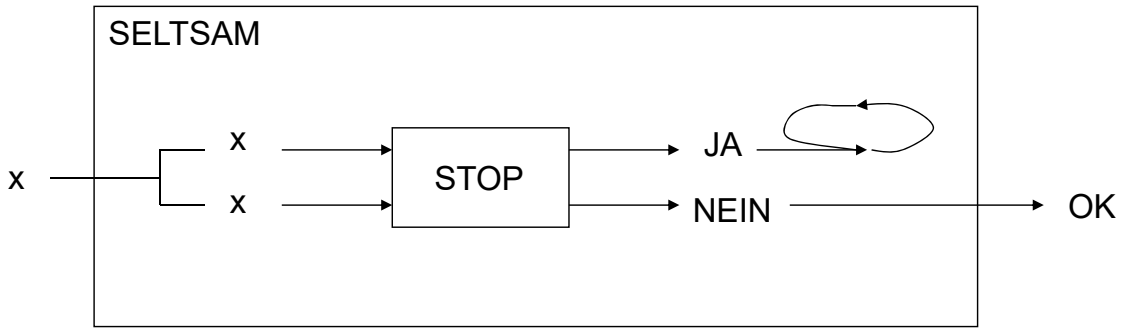
Veranschaulichung des Halteproblems

- Maschine (Algorithmus) `STOP` mit zwei Eingaben
  - Algorithmentext `x`
  - Eingabe `y` für `x`

  sowie zwei Ausgaben
  - `JA:` `x` stoppt bei Eingabe von `y`
  - `NEIN:` `x` stoppt nicht bei Eingabe von `y`
- Annahme: `STOP` löst (allgemeines) Halteproblem

Veranschaulichung des Halteproblems /2

- Konstruktion einer neuen Maschine `SELTSAM` mit `STOP`



Verhalten von SELTSAM

Hält SELTSAM bei der Eingabe von SELTSAM?

1. Wenn ja, so wird der JA-Ausgang von STOP angelaufen und SELTSAM gerät in die Endlosschleife, hält also nicht. Widerspruch!
2. Wenn nein, so wird der NEIN-Ausgang von STOP angelaufen und SELTSAM stoppt mit OK. Widerspruch!

Formaler Beweis I

- Annahme: $h : A^* \rightarrow A^*$ berechenbar

$$h(x) = \begin{cases} \epsilon, & \text{falls } x \in dom\ \varphi_x \\ a & \text{sonst} \end{cases}$$

- Da mit einem universellen Algorithmenmodell auch die folgende Funktion $g : A^* \rightarrow A^*$ notiert werden kann, ist auch sie berechenbar:

$$g(x) = \begin{cases} \varphi_x\,(x)a, & \text{falls } h\,(x) = \epsilon \\ \epsilon, & \text{sonst} \end{cases}$$

(Hinweis: Wenn h(x) = Є, dann ist sichergestellt, dass $\varphi_x\,(x)$ definiert ist. Aus diesem Grund terminiert g(x) in diesen Fällen und liefert als Ergebnis $\varphi_x\,(x)$a.)

- Offensichtlich gilt: $g(x) \neq \varphi_x\,(x)$ für alle $x \in A^*$.

Formaler Beweis II

- Es gibt also einen Algorithmus mit einem Text $y \in A^*$, der g berechnet, d. h., es gibt ein y mit $g = \varphi_y$.
  Damit folgt nun:

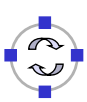
$$\varphi_y(y) = g(y) \neq \varphi_y(y)$$

- Dies ist offenbar ein Widerspruch, der sich nur dadurch lösen lässt, dass wir die Annahme aufgeben, h sei berechenbar.
- Erneut ein Diagonalbeweis!

Weitere nicht entscheidbare Probleme

Jede nichttriviale semantische Eigenschaft von Algorithmen ist nicht entscheidbar.

1. Ist die berechnete Funktion total? Überall undefiniert? Injektiv? Surjektiv? Bijektiv? etc. etc.
2. Berechnen zwei gegebene Algorithmen dieselbe Funktion?
3. Ist ein gegebener Algorithmus korrekt, d. h., berechnet er die gegebene (gewünschte) Funktion?
4. Auch einfache Fragestellungen können nichtentscheidbar sein: Post'sches Korrespondenzproblem



Post'sches Korrespondenzproblem

- Gegeben seien ein Alphabet A und zwei gleich lange Listen von Worten über A:

$$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$$
$$\beta = (\beta_1, \beta_2, \ldots, \beta_n)$$

wobei $\alpha_i, \beta_i \in A^+ = A^* - \{ \epsilon \}$ und $n \geq 1$.

- Das Problem besteht nun darin, eine „Korrespondenz" zu finden, d. h. eine endliche Folge $(i_1, i_2, \ldots, i_k)$, $i_j \in \{ 1, \ldots, n \}$ für $j = 1, 2, \ldots,$ k, so dass gilt:

$$\alpha_{i1}\ \alpha_{i2} \ldots \alpha_{ik} = \beta_{i1}\ \beta_{i2} \ldots \beta_{ik}$$

Post'sches Korrespondenzproblem: Bsp. 1

- Eingabelisten:

$A = \{ 0, 1 \}$ $\quad \alpha = (1\ ,\ 1\ 0\ 1\ 1\ 1\ ,\ 1\ 0)$
$\quad \beta = (1\ 1\ 1\ ,\ 1\ 0\ ,\ 0)$

- Dieses Post'sche Korrespondenzproblem besitzt eine Lösung, nämlich die Korrespondenz (2,1,1,3):

$$1\ 0\ 1\ 1\ 1\ .\ 1\ .\ 1\ .\ 1\ 0 = 1\ 0\ .\ 1\ 1\ 1\ .\ 1\ 1\ 1\ .\ 0$$

Post'sches Korrespondenzproblem: Bsp. 2 I

- Post'sches Korrespondenzproblem:

$$A = \{ 0, 1 \} \qquad \alpha = (1\,0\,,\, 0\,1\,1\,,\, 1\,0\,1)$$
$$\beta = (1\,0\,1\,,\, 1\,1\,,\, 0\,1\,1)$$

- Keine Lösung!

Post'sches Korrespondenzproblem: Bsp. 2 II

- Gäbe es eine Lösung, so müsste sie mit 1 anfangen:

$$1\,0\ldots \stackrel{?}{=} 1\,0\,1\ldots$$

- Dann ein i mit $\alpha_i = 1\ldots$, also 1 oder 3.
  - (1, 1, . . .) ergibt

$$1\,0\,1\,0\ldots \neq 1\,0\,1\,1\,0\,1\ldots$$

  - (1, 3, . . .) ergibt

$$1\,0\,1\,0\,1\ldots \stackrel{?}{=} 1\,0\,1\,0\,1\,1\ldots$$

- Index 3 muss wieder gewählt werden, (1, 3, 3, . . .) ergibt

$$1\,0\,1\,0\,1\,1\,0\,1\ldots \stackrel{?}{=} 1\,0\,1\,0\,1\,1\,0\,1\,1\ldots$$

- Wieder müsste 3 gewählt werden usw. usw. usw.
- Terminiert nicht!

Korrektheit

- Korrektheit als wichtige Eigenschaft
- Motivierende Beispiele: siehe Kapitel 1
- Problem: im Allgemeinen nicht entscheidbar
- Daher:
  - Pragmatisches Testen
  - Beweisführung im Einzelfall

Korrektheit von Algorithmen

- Nachweis der Korrektheit eines Algorithmus bezieht sich immer auf eine *Spezifikation* dessen, was er tun *soll*
- Daher *relative* Korrektheit
  - *Spezifikation*: eindeutige Festlegung der berechneten Funktion bzw. des Terminierungsverhaltens
  - *Verifikation*: formaler Beweis der Korrektheit bezüglich einer formalen Spezifikation
  - *Validation*: (nicht-formaler) Nachweis der Korrektheit bezüglich einer informellen oder formalen Spezifikation (etwa systematisches Testen)

Angabe von Vor- und Nachbedingungen

$$\{\ \text{VOR}\ \}\ \ \text{ANW}\ \ \{\ \text{NACH}\ \} \qquad (*)$$

- VOR und NACH sind dabei Aussagen über den Zustand vor bzw. nach Ausführung der Anweisung ANW
- Aussage bedeutet:

> *Gilt VOR unmittelbar vor Ausführung von ANW und terminiert ANW, so gilt NACH unmittelbar nach Ausführung von ANW.*

  - Terminiert ANW nicht, so ist (∗) trivialerweise wahr, wie auch immer VOR und NACH aussehen!
  - Auch ist (∗) trivialerweise wahr, wenn VOR nicht gilt, gleichgültig, ob ANW terminiert oder nicht und ob NACH gilt oder nicht!

Partielle versus totale Korrektheit

```
PROG:     var X,Y, ...: int ;
              P,Q ...: bool ;
          input X_1, . . . , X_n;
                α ;
          output Y_1, . . . , Y_m
```

- PROG heißt partiell korrekt bzgl. VOR und NACH genau dann wenn $\{\ VOR\ \}\alpha\{\ NACH\ \}$ wahr ist.
- PROG heißt total korrekt bzgl. VOR und NACH genau dann wenn PROG partiell korrekt ist bzgl. VOR und NACH, und wenn α darüber hinaus immer dann terminiert, wenn vorher VOR gilt.

Beispiele für Vor- und Nachbedingungen

- $\{ X = 0 \}\ X := X + 1\ \{ X = 1 \}$
  ist wahr
- $\{ \mathbf{true} \}\ X := Y\ \{ X = Y \}$
  ist ebenfalls wahr
- $\{ Y = a \}\ X := Y\ \{ X = a \wedge Y = a \}$
  ist wahr für alle $a \in \mathbb{Z}$
- $\{ X = a \wedge Y = b \wedge a \neq b \}\ X := Y;\ Y := X\ \{ X = b \wedge Y = a \}$
  ist *falsch* für alle $a, b \in \mathbb{Z}$
  (ein beliebter „Anfängerfehler“ beim Programmieren eines Werte-Tausches zweier Variablen)
- $\{ X = a \wedge Y = b \}\ Z := X;\ X := Y;\ Y := Z\ \{ X = b \wedge Y = a \}$
  ist wahr für alle $a, b \in \mathbb{Z}$
  (die korrekte Version des Werte-Tausches)

Beispiele für Vor- und Nachbedingungen

- $\{ \mathbf{false} \}$ `ANW` $\{$ `NACH` $\}$
  ist wahr für alle `ANW` und `NACH`
- $\{ \mathbf{true} \}$ `ANW` $\{ \mathbf{false} \}$
  ist genau dann wahr, wenn `ANW` nicht terminiert
- $\{ X = a \}$ **while** $X \neq 0$ **do** $X := X - 1$ **od** $\{ X = 0 \}$
  ist wahr für alle $a \in \mathbb{Z}$ – insbesondere auch für $a < 0$, da dann die **while**-Schleife nicht terminiert

Beweise für atomare Anweisungen

- $X := t$: jedes Auftreten von t in *VOR* durch X ersetzen

  $\{ 3 > 0 \}\ X := 3\ \{ X > 0 \}$

  auch bei X auf beiden Seiten:

  $\{ X + 1 > 0 \}\ X := X + 1\ \{ X > 0 \}$

- Hilfskonstruktion: „neue“ Werte von X als X' markieren

  $\{ X + 1 > 0 \}\ X' := X + 1\ \{ X' > 0 \}$

- Umformungen um Ersetzung zu ermöglichen:

  $\{ X > 0 \}\ X' := X - 1\ \{ X' \geq 0 \}$

- Äquivalenzen $(X > 0) \equiv ((X - 1) + 1 > 0)$ und $(X \geq 0) \equiv (X + 1 > 0)$ ausnutzen:

  $\{ X > 0 \}\ X := X - 1\ \{ X \geq 0 \}$

  $\{ (X - 1) + 1 > 0 \}\ X' := X - 1\ \{ X' + 1 > 0 \}$

Beweise bei Sequenzen

- Beweisschritt in zwei Schritte aufteilen

  $\{ \texttt{VOR} \}\ \alpha; \beta\ \{ \texttt{NACH} \}$

- Zwischenbedingung `MITTE` finden, dann

  $\{ \texttt{VOR} \}\ \alpha\ \{ \texttt{MITTE} \}$

  und

  $\{ \texttt{MITTE} \}\ \beta\ \{ \texttt{NACH} \}$

  zeigen

Beweise bei Selektion

- Fallunterscheidung notwendig

$$\{\ \mathrm{VOR}\ \}\ \textbf{if}\ B\ \textbf{then}\ \alpha\ \textbf{else}\ \beta\ \textbf{fi}\ \{\ \mathrm{NACH}\ \}$$

- Teilbeweise

$$\{\ \mathrm{VOR} \wedge B\ \}\ \alpha\ \{\ \mathrm{NACH}\ \}$$

und

$$\{\ \mathrm{VOR} \wedge \neg B\ \}\ \beta\ \{\ \mathrm{NACH}\ \}$$

- Wichtig ist, dass die Nachbedingung gilt, egal welcher Zweig selektiert wurde

Beweise bei Schleifen

- *Schleife* der Form

$$\{\ \mathrm{VOR}\ \}\ \textbf{while}\ B\ \textbf{do}\ \beta\ \textbf{od}\ \{\ \mathrm{NACH}\ \}$$

- Zunächst geeignete *Schleifeninvariante* P finden, dann:

1. Prüfen, ob

$$\mathrm{VOR} \Rightarrow P$$

2. Schleifenrumpf bewahrt P:

$$\{\ P \wedge B\ \}\ \beta\ \{\ P\ \}$$

3. Nachbedingung muss nach Verlassen der Schleife gelten

$$\{\ P \wedge \neg B\ \} \Rightarrow \mathrm{NACH}$$

Korrektheit von MULT I

```
MULT:    var W,X,Y,Z : int;
         input X, Y
         Z:=0;
         W:=Y;
         while W ≠ 0 do Z:=Z+X; W:=W-1 od;
         output Z
```

Der Schleifenrumpf sei wieder mit β bezeichnet.

- Vorbedingung: $\{ Y \geq 0 \}$
- Nachbedingung: $\{ Z = X \cdot Y \}$

Korrektheit von MULT II

Schleifeninvariante P

$$P = (X \cdot Y = Z + W \cdot X)$$

- Schleifeninvariante beim Eintritt in die Schleife:

  $\{ Y \geq 0 \}\ Z := 0;\ W := Y\ \{ X' \cdot Y' = Z' + W' \cdot X' \}$

  neue Werte durch alte Werte gemäß der Zuweisungen ersetzen:

  $\{ X \cdot Y = X' \cdot Y' = Z' + W' \cdot X' = 0 + Y \cdot X \}$

  P gilt vor dem ersten Schleifendurchlauf!

Korrektheit von MULT III

Schleifeninvariante $P = (X \cdot Y = Z + W \cdot X)$

- Wenn $P \wedge B$ vor Rumpf gilt, muss P auch danach gelten:

$$\{ X \cdot Y = Z + W \cdot X \wedge W \neq 0 \}$$
$$Z := Z + X;\ W := W - 1$$
$$\{ X' \cdot Y' = Z' + W' \cdot X' \}$$

- Als neu markierte Variablen ersetzen:

$$X \cdot Y = X' \cdot Y' = Z' + W' \cdot X'$$
$$= (Z + X) + (W - 1) \cdot X$$
$$= Z + X + W \cdot X - X$$
$$= Z + W \cdot X$$

- Der Rumpf erhält die Schleifeninvariante P.

Korrektheit von MULT IV

Schleifeninvariante P

$$P = (X \cdot Y = Z + W \cdot X)$$

- Es bleibt als dritter Schritt nur noch die Aufgabe, zu zeigen, dass $(P \wedge \neg B)$ die ursprüngliche Nachbedingung $Z = X \cdot Y$ impliziert:

$$(P \wedge \neg B) \equiv (X \cdot Y = Z + W \cdot X \wedge W = 0) \equiv (X \cdot Y = Z)$$

Korrektheit imperativer Algorithmen am Beispiel

```
XYZ:    var W,X,Y,Z : int ;
        input X
        Z:=0; W:=1; Y:=1 ;
        while W ≤  X
              do Z:=Z+1; W:=W+Y+2; Y:=Y+2 od;
        output Z.
```

$$VOR \equiv (X \geq 0)$$

$$NACH \equiv (Z^2 \leq X < (Z+1)^2) \equiv (Z = \lfloor \sqrt{X} \rfloor).$$

Schleifeninvariante

$\alpha = (Z := 0;\quad W := 1;\quad Y := 1)$

$\beta = (Z := Z + 1;\quad W := W + Y + 2;\quad Y := Y + 2)$

$P = (Z^2 \leq X \;\wedge\; (Z+1)^2 = W \;\wedge\; 2 \cdot Z + 1 = Y \;\wedge\; Y > 0)$

P ist die “Zwischenbedingung”, von der wir zeigen:

- dass sie am Anfang jeden Durchlaufs durch die **`while`**-Schleife gilt (die “Schleifeninvariante”), und
- dass sie nach Ablauf der **`while`**-Schleife die Nachbedingung NACH impliziert.

Beweisvorgehen für die Schleifeninvariante:

1. $\{ VOR \}\ \alpha\ \{ P \}$
2. $\{ P \wedge W \leq X \}\ \beta\ \{ P \}$
3. $P \wedge W > X \Rightarrow NACH$

Beweis der partiellen Korrektheit

zu 1. Mit den Werten $Z = 0$, $W = Y = 1$ nach α gilt dort P, *da:*
$(0^2 \leq X \wedge (0+1)^2 \leq 1 \wedge 2 \cdot 0 + 1 = 1 \wedge 1 > 0)$

zu 2. Offenbar gilt:

a.) $\{ (Z + 1)^2 = W \wedge W \leq X \} \; \beta \; \{ (Z')^2 \leq X \}$

b.) $\{ (Z + 1)^2 = W \wedge 2 \cdot Z + 1 = Y \} \; \beta$

$\{ W' = W + Y + 2 = (Z + 1)^2 + 2 \cdot Z + 1 + 2$

$= Z^2 + 2 \cdot Z + 1 + 2 \cdot Z + 1 + 2$

$= Z^2 + 4 \cdot Z + 4 = (Z + 2)^2 = (Z + 1 + 1)^2 = (Z' + 1)^2 \}$

c.) $\{ 2 \cdot Z + 1 = Y \} \; \beta$

$\{ Y' = Y + 2 = 2 \cdot Z + 1 + 2 = 2 \cdot (Z + 1) + 1 = 2 \cdot Z' + 1 \}$

d.) $\{ Y > 0 \} \; \beta \; \{ Y' = Y + 2 > 0 \}$

Hieraus folgt insgesamt $\{ P \wedge W \leq X \} \; \beta \; \{ P \}$

zu 3. $P \wedge W > X \Rightarrow Z^2 \leq X \wedge X < (Z + 1)^2 \Rightarrow NACH$

Beweis der Terminierung

Für den Wert von $u = X - W$ gilt:

1. $W \leq X \Rightarrow u \geq 0$, d. h. u bleibt bei allen Schleifendurchläufen nichtnegativ.
2. u wird bei jedem Schleifendurchlauf echt kleiner.

zu 1. Dies ist offensichtlich.

zu 2. Sei u der Wert unmittelbar vor Ablauf von β, und sei u´ der Wert unmittelbar danach. Dann gilt:

$u = X - W$

$u' = X - (W + Y + 2) = X - W - Y - 2$

Da $Y > 0$ ist (!), ist u´ < u.

Korrektheit applikativer Algorithmen

- Für den Nachweis von Eigenschaften — wie z. B. der Korrektheit — *applikativer* Algorithmen verwendet man typischerweise Induktionsbeweise, die der Struktur der rekursiven Funktionsdefinition folgen.
- Mc'Carthy's 91-Funktion:

$$f(x) = \textbf{if } x > 100 \textbf{ then } x - 10 \textbf{ else } f(f(x + 11)) \textbf{ fi}.$$

Sei g(x) = **if** x > 100 **then** x - 10 **else** 91 **fi**.

- Wir beweisen: $f(x) = g(x)$ für alle $x \in \mathbb{Z}$.

Beweis $f(x) = g(x)$

- Induktionsanfang: Für alle $x > 100$ ist $f(x) = x - 10 = g(x)$.
  Die Induktion verläuft "rückwärts", d. h. wir zeigen:
  Gilt $f(y) = g(y)$ für alle $y \geq x$, so gilt auch $f(x - 1) = g(x - 1)$; also:
- Induktionsannahme: Es gelte $f(y) = g(y)$ für $y \geq x$
- Induktionsschritt:
  - Fall: Sei $101 > x \geq 91$. Dann gilt:
    $f(x - 1) = f(f(x - 1 + 11)) = f(f(x + 10)) =$
    $f(x + 10 - 10) = f(x) \overset{IA}{=} g(x) = 91 = g(x - 1)$
  - Fall: Sei $90 \geq x$. Dann gilt: $f(x - 1) = f(f(x + 10))$
    Aufgrund der Induktionsannahme ist
    $f(x + 10) = g(x + 10) = 91$, also folgt:
    $f(x - 1) = f(91) = g(91) = 91 = g(x - 1)$

Dies beweist $f(x) = g(x)$ für alle $x \in \mathbb{Z}$.

Komplexität

- Korrektheit eines Algorithmus ist unerlässlich
- Wünschenswert: möglichst geringer Aufwand
- Daher:
  - Abschätzung des Aufwands von Algorithmen (Komplexität)
  - Mindestaufwand zur Lösung von Problemen dieser Klasse

Motivierendes Beispiel

- Sequenzielle Suche in Folgen
- Gegeben:
  - eine Zahl $n \geq 0$,
  - n Zahlen $a_1, \ldots, a_n$, alle verschieden
  - eine Zahl b
- Gesucht:
  - Index $i = 1, 2, \ldots, n$, so dass $b = a_i$ falls Index existiert
  - sonst $i = n + 1$
- Lösung

$$i := 1; \textbf{while } i \leq n \wedge b \neq a_i \textbf{ do } i := i + 1 \textbf{ od}$$

Aufwand der Suche

- Ergebnis hängt von der Eingabe ab, d. h. von n, a_1, . . . , a_n und b:
  1. *erfolgreiche* Suche (wenn $b = a_i$): $S = i$ Schritte
  2. *erfolglose* Suche: $S = n + 1$ Schritte

Aufwand der Suche /2

- Ziel: globalere Aussagen, die nur von *einer* einfachen Größe abhängen, z. B. von der Länge n der Folge
- Fragen:
  - A: Wie groß ist S für gegebenes n *im schlechtesten Fall?*
  - B: Wie groß ist S für gegebenes n *im Mittel?*

Analyse für erfolgreiche Suche (A)

- Im schlechtesten Fall:
  b wird erst im letzten Schritt gefunden: $b = a_n$

$S = n$ im schlechtesten Fall

Analyse für erfolgreiche Suche (B)

- Im Mittel:
  - Wiederholte Anwendung mit verschiedenen Eingaben
  - Annahme über Häufigkeit
    Wie oft wird *b* an erster, zweiter, . . . , letzter Stelle gefunden?
  - Gleichverteilung

Läuft der Algorithmus N-mal, $N >> n$, so wird *b* gleich häufig an erster, zweiter . . . , letzter Stelle gefunden, also jeweils $\frac{N}{n}$-mal.

Analyse für erfolgreiche Suche (B) /2

- Insgesamt für *N* Suchvorgänge:

$$M = \frac{N}{n} \cdot 1 + \frac{N}{n} \cdot 2 + \ldots + \frac{N}{n} \cdot n$$

$$= \frac{N}{n}(1 + 2 + \ldots + n) = \frac{N}{n} \cdot \frac{n(n+1)}{2}$$

$$= N \cdot \frac{n+1}{2}$$

- für **eine** Suche: $S = \frac{M}{N}$ Schritte, also

$S = \frac{n+1}{2}$ im Mittel (bei Gleichverteilung)

Asymptotische Analyse

- Analyse der Komplexität ⇝ Angabe einer Funktion

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

als Maß für Aufwand

- Bedeutung: $f(n) = a$
  - bei Problem der Größe n
  - beträgt der Aufwand a
- Problemgröße: Umfang der Eingabe, wie z. B. Anzahl der zu sortierenden oder zu durchsuchenden Elemente
- Aufwand: Rechenzeit (Abschätzung als Anzahl der Operationen wie z. B. Vergleiche), Speicherplatz

Aufwand für Schleifen

- Wie oft wird die Wertzuweisung „$x := x + 1$“ in folgenden Anweisungen ausgeführt?

1.	$x := x + 1$	1mal
2.	**for** $i := 1$ **to** n **do** $x := x + 1$ **od**	n-mal
3.	**for** $i := 1$ **to** n **do** **for** $j := 1$ **to** n **do** $x := x + 1$ **od od**	n^2 -mal

Aufwandsfunktion

- Aufwandsfunktion $f : \mathbb{N} \rightarrow \mathbb{N}$ meist nicht exakt bestimmbar, daher:
  - Abschätzung des *Aufwandes im schlechtesten Fall*
  - Abschätzung des *Aufwandes im Mittel*

  und „ungefähres Rechnen in Größenordnungen“

O-Notation

- Angabe der asymptotischen oberen Schranke für Aufwandsfunktion ⇝ Wachstumsgeschwindigkeit bzw. Größenordnung
- Asymptote: Gerade, der sich eine Kurve bei immer größer werdender Entfernung vom Koordinatenursprung unbegrenzt nähert
- Einfache Vergleichsfunktion $g : \mathbb{N} \rightarrow \mathbb{N}$ für Aufwandsfunktion f mit

$$f(n) = O(g(n))$$

O-Notation /2

- Definition

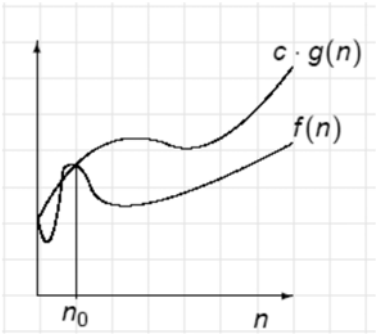
$$f(n) = O(g(n)) : \Leftrightarrow \exists c, n_0 \; \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

- $\frac{f(n)}{g(n)}$ ist für genügend große n durch eine Konstante c beschränkt, d. h.

> f wächst nicht schneller als g

- Um im konkreten Fall für gegebene Funktionen f und g zu zeigen, dass $f(n) = O(g(n))$ gilt, müssen jeweils ein geeignetes n_0 und c angegeben werden.

O-Notation /3



- Eigentlich $f \in O(g)$ statt $f = O(g)$, da f zur Klasse der Funktionen gehört, deren Wachstum asymptotisch durch g beschränkt ist:

  $O(g) = \{ f \mid \exists\, c, n_0 : \forall\, n \geq n_0 : f(n) \leq c \cdot g(n) \}$

- Da $O(g)$ **obere Grenze** für eine Klasse von Funktionen ist: Vereinfachung möglich ⇝ **Rechnen in Größenordnungen**

Rechnen in Größenordnungen

- Weglassen von multiplikativen Konstanten

  $2 \cdot n = O(n)$

- Basis des Logarithmus ist unerheblich

  $\log_2 n = O(\log n)$

  Grund: Basiswechsel entspricht Multiplikation von Konstanten, da

  $\log_b n = \log_a n \cdot \log_b a$

- Beschränkung auf höchsten Exponenten

  $n^2 + 3n - 3 = O(n^2)$

- $n^2 + 3n - 3 \leq c \cdot n^2$ ⇝ $1 + \frac{3}{n} - \frac{3}{n^2} \leq c$ (z. B. für $c = 2$ und $n = 1$)

Die Ω-Notation und die Θ-Notation

- Die O-Notation ist nützlich für die Abschätzung durch *obere Schranken*, da gezeigt wird, dass eine Funktion *f*, die eine zu untersuchende Größe (z.B. Laufzeit oder Speicherbedarf) beschreibt, asymptotisch langsamer wächst als eine Funktion *g*

- Oft müssen jedoch auch *untere Schranken* abgeschätzt werden („Mindestbedarf“):
  - Hierfür definiert man analog die Ω-Notation:

    $\Omega(g) = \{ f \mid \exists\, c, n_0 : \forall\, n \geq n_0 : f(n) \geq c \cdot g(n) \}$

- Oder man möchte eine *„scharfe“ asymptotische Schranke*:
  - Hierfür definiert man entsprechend die Θ-Notation:

    $\Theta(g) = \{ f \mid \exists\, c_1, c_2, n_0 : \forall\, n \geq n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \}$

Komplexitätsklassen

$O(1)$	konstanter Aufwand
$O(\log n)$	logarithmischer Aufwand
$O(n)$	linearer Aufwand
$O(n \cdot \log n)$	
$O(n^2)$	quadratischer Aufwand
$O(n^k)$ für ein $k \geq 0$	polynomialer Aufwand
$O(2^n)$	exponentieller Aufwand

Wachstum in Abhängigkeit von Problemgröße

$f(n)$	$n = 2$	$2^4 = 16$	$2^8 = 256$	$2^{10} = 1024$	$2^{20} = 1048576$
$\text{ld}n$	1	4	8	10	20
n	2	16	256	1024	1048576
$n \cdot \text{ld}n$	2	64	2048	10240	20971520
n^2	4	256	65536	1048576	$\approx 10^{12}$
n^3	8	4096	16777200	$\approx 10^9$	$\approx 10^{18}$
2^n	4	65536	$\approx 10^{77}$	$\approx 10^{308}$	$\approx 10^{315653}$

- Zwei Vergleichswerte:
  - Geschätztes Alter unseres Sonnensystems: $\sim 2 \cdot 10^{17}$ s
  - Geschätzte Anzahl Elektronen im Universum: $\sim 8.37 \cdot 10^{77}$

Berechenbare Problemgrößen bei vorgegebener Zeit

G	T = 1 min.	1.Std.	1 Tag	1 Woche	1 Jahr
n	$6 \cdot 10^7$	$3{,}6 \cdot 10^9$	$8{,}6 \cdot 10^{10}$	$6 \cdot 10^{11}$	$3 \cdot 10^{13}$
n^2	7750	$6 \cdot 10^4$	$2.9 \cdot 10^5$	$7.8 \cdot 10^5$	$5{,}6 \cdot 10^6$
n^3	391	1530	4420	8450	31600
2^n	25	31	36	39	44

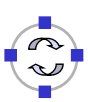
Annahme: 1 Rechenschritt benötigt 1 Mikrosekunde
(10^{-6} s $\Rightarrow$ 10^6 Rechenschritte/s)

Typische Problemklassen

Aufwand	Problemklasse
$O(1)$	einige Suchverfahren für Tabellen (Hashing)
$O(\log n)$	allgemeine Suchverfahren für Tabellen (Baum-Suchverfahren)
$O(n)$	sequenzielle Suche, Suche in Texten, syntaktische Analyse von Programmen (bei „guter“ Grammatik)
$O(n \cdot \log n)$	Sortieren

Typische Problemklassen /2

Aufwand	Problemklasse
$O(n^2)$	einige dynamische Optimierungsverfahren (z. B. optimale Suchbäume), Multiplikation Matrix-Vektor (einfach)
$O(n^3)$	Matrizen-Multiplikation (einfach)
$O(2^n)$	viele Optimierungsprobleme (z. B. optimale Schaltwerke), automatisches Beweisen (im Prädikatenkalkül 1. Stufe)



P- vs. NP-Probleme

- Grenze zwischen **effizient** lösbaren (polynomialer Aufwand) und **nicht effizient** lösbaren (exponentieller Aufwand) Problemen
- **Problemklasse P**: Menge aller Probleme, die mit Hilfe *deterministischer* Algorithmen in polynomialer Zeit gelöst werden können
- **Problemklasse NP**: Menge aller Probleme, die nur mit Hilfe *nichtdeterministischer* Algorithmen in polynomialer Zeit gelöst werden können
  - Nichtdeterminismus: „Raten“ der richtigen Variante bei mehreren Lösungen
  - Umsetzung mit deterministischen Algorithmen: exponentieller Aufwand

Erfüllbarkeitsproblem

- Problem der Klasse NP
  - Gegeben: aussagenlogischer Ausdruck mit Variablen *a*, *b*, *c* etc., logischen Operatoren Λ , V , ¬ und Klammern
  - Gesucht: Algorithmus der prüft, ob Formel erfüllbar ist, d. h., ob Belegung der Variablen mit booleschen Werten **`true`** und **`false`** existiert, so dass Formel **`true`** liefert
- Lösungsidee: Testen aller Belegungen (nicht effizient)

NP-Vollständigkeit

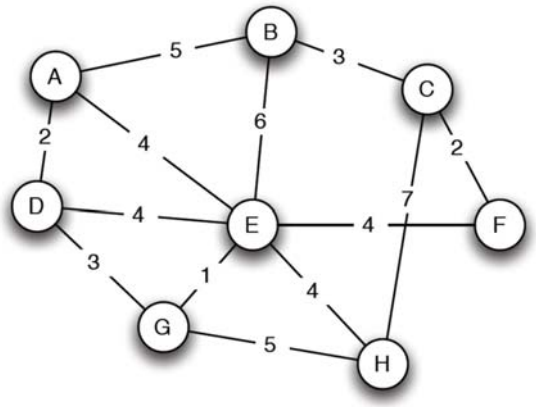
- Allgemein: $P \subseteq NP$
- Jedoch offenes Problem, ob sich NP-Probleme nicht doch mit polynomialen Aufwand lösen lassen (also $P = NP$)
- Beweisbar: Existenz einer Klasse von verwandten Problemen aus NP mit folgender Eigenschaft

> Falls eines dieser Probleme in polynomialer Zeit mit einem deterministischen Algorithmus gelöst werden könnte, so ist dies für alle Probleme aus NP möglich.

⇝ **NP-vollständige** Probleme

Problem des Handlungsreisenden

- Problem der Graphentheorie (bekannt als „*Traveling Salesman*“)
- Gegeben: Graph mit n Knoten (Städten) und m Kanten (Straßen); Straßen sind mit Entfernung versehen



- Gesucht: kürzeste Route, die alle Städte besucht und an den Startpunkt zurückkehrt

Rucksackproblem

- Wanderer will seinen Rucksack mit verschiedenen Gegenständen packen (auch *knapsack problem*)
- Gegeben: Rucksack mit Kapazität C; n Gegenstände mit jeweils Gewicht g_i und Wert w_i,
- Gesucht: Auswahl der Gegenstände (Indexierung $I \subseteq \{ 1, \ldots, n \}$), das Gesamtgewicht die Kapazität nicht überschreitet

$$\sum_{i \in I} g_i \leq C$$

und Summe der Werte maximal ist

$$\sum_{i \in I} w_i \text{ ist maximal}$$

Analyse von Algorithmen

- Bestimmung der Laufzeitkomplexität von Algorithmen in *O*-Notation
- Abschätzung der Größenordnung durch einfache Regeln

`for`-Schleifen

- Aufwand

  Laufzeit := *max. Laufzeit der inneren Anweisung* ·*Anzahl Iterationen*

- Beispiel

```
for i := 1 to n do
    a[i] := 0;
od
```

wenn innere Operation $O(1)$, dann $n \cdot O(1) = O(n)$

Geschachtelte `for`-Schleifen

- Aufwand

  Laufzeit :=

  Laufzeit der inneren Anweisung ·*Produkt der Größen aller Schleifen*

- Beispiel

```
for i := 1 to n do
    for j := 1 to n do
        k := k + 1;
    od
od
```

- $n \cdot n \cdot O(1) = O(n^2)$

Sequenz

- Beispiel

```
for i := 1 to n do
    a[i] := 0;
od;

for i := 1 to n do
    for j := 1 to n do
        a[i] := a[i] + a[j] + i + j ;
    od
od
```

- $O(n) + O(n^2) = O(n^2)$

if-else-Bedingungen

- Aufwand

  Laufzeit := *Aufwand für Test* +max *(Aufwand für* A_1 , *Aufwand für* A_2)

- Beispiel

```
if x > 100 then
    y := x;
else
    for i := 1 to n do
        if a[i] > y then y := a[i] fi
    od
fi
```

- $O(n)$

Zusammenfassung

- Berechenbarkeit und Entscheidbarkeit, Halteproblem
- Korrektheit von Algorithmen
  - Spezifikation gewünschter Eigenschaften
  - Vor- und Nachbedingungen: Korrektheit imperativer Algorithmen
  - Induktionsbeweise: Korrektheit applikativer Algorithmen
- Komplexität
  - *O*-Notation
  - Komplexitätsklassen
  - P- vs. NP-Probleme
- Literatur: Saake/Sattler: *Algorithmen und Datenstrukture*n, Kapitel 7

Anhang 1: Vollständige Induktion

- Das Beweisverfahren der *vollständigen Induktion* ist ein Verfahren, mit dem Aussagen über natürliche Zahlen bewiesen werden können.
- Neben Aussagen über natürliche Zahlen können damit auch gut Aussagen bewiesen werden, die
  - rekursiv definierte Strukturen und
  - abzählbare Strukturen

  betreffen.
- Grundidee – Eine Aussage ist gültig für alle natürlichen Zahlen $n \in \mathbb{N}$, wenn man nachweisen kann:
  - Die Aussage gilt für die erste natürliche Zahl $n = 1$ (Induktionsanfang).
  - Wenn die Aussage für eine natürliche Zahl n gilt, dann gilt sie auch für ihren Nachfolger $n+1$. (Induktionsschritt).

Einführendes Beispiel 1

- Gegeben sei die folgende rekursiv definierte Folge:
  - $a_1 = 1$ und $a_{n+1} = 2 + a_n$
- Vermutung: $a_n = 2 \cdot n - 1$
- Beweis:

  Induktionsanfang für k = 1:
  - $a_1 = 2 \cdot 1 - 1 = 2 - 1 = 1$ ✓

  Induktionsschritt:
  - Induktionsvoraussetzung: $a_k = 2 \cdot k - 1$
  - Induktionsschritt, zu zeigen: $a_{k+1} = 2 \cdot (k+1) - 1$

    $a_{k+1} = 2 + a_k = 2 + 2 \cdot k - 1 = 2 \cdot (k+1) - 1$ ✓

Einführendes Beispiel 2 (Summenformel)

Folge: $S(n) = \sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = \frac{1}{2} \cdot n \cdot (n+1)$

Beweis:

Induktionsanfang: $\frac{1}{2} \cdot 1 \cdot (1+1) = \frac{1}{2} \cdot 2 = 1$ ✓

Induktionsschritt:

Induktionsvoraussetzung: A(k) gilt, d. h. $\sum_{i=1}^{k} i = \frac{1}{2} \cdot k \cdot (k+1)$

zu zeigen: dann gilt auch A(k+1), d. h. $\sum_{i=1}^{k+1} i = \frac{1}{2} \cdot (k+1) \cdot (k+2)$

$$\sum_{i=1}^{k+1} i = \left(\sum_{i=1}^{k} i \right) + (k+1) = \frac{1}{2} \cdot k \cdot (k+1) + (k+1)$$

$$= \frac{1}{2} \left(k^2 + k + 2 \cdot k + 2 \right) = \frac{1}{2} \cdot (k+2) \cdot (k+1)$$ ✓

Einführendes Beispiel 3 (a)

- Betrachte die folgende Summenformel:

$$T(n) = \sum_{k=1}^{n} 3 + 5k = 8 + 13 + 18 + 23 + \cdots + (3 + 5n)$$

- Da $S(n) = n^2/2 + n/2$ und bei *T(n)* jedes Element etwas mehr als fünfmal (und damit konstant) größer ist als sein Vorgänger, ist es plausibel anzunehmen, dass auch *T(n)* durch einen quadratischen Ausdruck angegeben werden kann:

$$T(n) = c_1 n^2 + c_2 n + c_3$$

- Da *T(0) = 0* ist, muss $c_3 = 0$ gelten
- Weiterhin: $1 \cdot c_1 + 1 + c_2 = 8$

$$4 \cdot c_1 + 2 \cdot c_2 = 13 + 8$$

- Lösen des Gleichungssystems ergibt $c_1 = 2.5$ und $c_2 = 5.5$
- Somit vermuten wir: $T(n) = 2.5n^2 + 5.5n$

Einführendes Beispiel 3 (b)

- Unsere bisherige Rechnung beruht auf einer Annahme, die noch nicht bewiesen ist!
- Daher muss noch gezeigt werden, dass der gewählte Ansatz tatsächlich korrekt ist:

Induktionsanfang: n = 1: $T(1) = 5 + 3 = 2.5 \cdot 1^2 + 5.5 \cdot 1$ ✓

Induktionsschritt: n ⇝ n + 1:

$$T(n+1) = T(n) + 5(n+1) + 3$$

$$= 2.5n^2 + 5.5n + 5n + 8$$

$$= 2.5n^2 + 5n + 2.5 + 5.5n + 5.5$$

$$= 2.5(n+1)^2 + 5.5(n+1)$$ ✓

Einführendes Beispiel 4

- Wenn n eine natürliche Zahl und 1 + x > 0 ist, dann gilt:

$$(1+x)^n \geq 1+nx$$

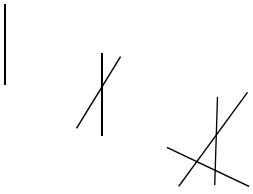
Induktionsanfang: n = 1: $1+x \geq 1+x$ ✓

Induktionsschritt: n ⇝ n + 1: $(1+x)^{n+1} \geq 1+(n+1)x$

$$(1+x)^{n+1} = (1+x)(1+x)^n \geq (1+x)(1+nx)$$
$$= 1+(n+1)x+nx^2 \geq 1+(n+1)x$$ ✓

Zählen von Regionen in einer Ebene (1)

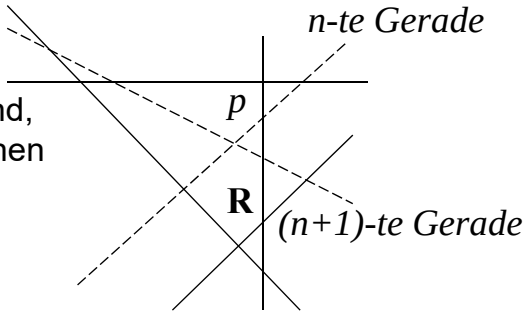
- Definition:
  Eine Menge von Geraden in einer Ebene befindet sich in *allgemeiner Position* genau dann wenn keine Geraden zueinander parallel verlaufen und sich in keinem Punkt mehr als zwei Geraden schneiden.
- Frage: Wieviele Regionen werden durch *n* Geraden in allgemeiner Position gebildet?
  - Für n = 1 ergeben sich zwei Regionen
  - Für n = 2 ergeben sich vier Regionen
  - Für n = 3 ergeben sich sieben Regionen



  → Für i ≤ 3, fügt die *i*-te Gerade offenbar *i* Regionen hinzu
- Behauptung: Wenn zu *(n-1)* vorhandenen Geraden in allgemeiner Position eine Gerade hinzugefügt wird, so dass sich die Geraden weiterhin in allgemeiner Position befinden, erhöht sich die Anzahl der Regionen um *n*.

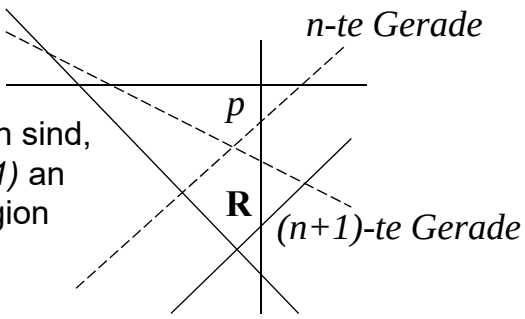
Zählen von Regionen in einer Ebene (2)

- Es bezeichne T(n) die Anzahl der gebildeten Regionen
- Dann folgt aus der Behauptung: $T(n) = 1 + \sum_{i=1}^{n} i = 1 + \frac{n \cdot (n+1)}{2}$
- Induktionsanfang: Die 2-te Gerade fügt offensichtlich 2 Regionen hinzu ✓
- Induktionsschritt: $n \rightsquigarrow n+1$
  - Wie erhöht das Hinzukommen einer Geraden die Anzahl der Regionen?
  - Da die Geraden in allgemeiner Position sind, kann die neue Gerade existierende Regionen nicht nur am Rand berühren
  - Daher werden existierende Regionen entweder in zwei Regionen geteilt oder die neue Gerade ist außerhalb der Region



Zählen von Regionen in einer Ebene (3)

- Induktionsschritt (Fortsetzung):
  - Wir müssen somit beweisen, dass die hinzukommende Gerade insgesamt n+1 neue Regionen entstehen lässt
  - Betrachten wir für einen Moment die Situation, dass die *n*-te Gerade nicht vorhanden ist:
    - In diesem Fall würde die *(n+1)*-te Gerade (die jetzt die *n*-te Gerade wäre) laut Induktionshypothese *n* Regionen erzeugen
    - Es reicht somit zu zeigen, dass die Anwesenheit der n-ten Geraden dafür sorgt, dass die *(n+1)*-te Gerade eine weitere Region erzeugt
  - Fügen wir die n-te Gerade wieder hinzu:
    - Da alle Geraden in allgemeiner Position sind, schneiden sich die Geraden *n* und *(n+1)* an einem Punkt *p*, der innerhalb einer Region *R* liegen muss

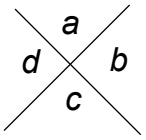


Zählen von Regionen in einer Ebene (4)

- Induktionsschritt (Fortsetzung):
  - Fügen wir die *n*-Gerade wieder hinzu (Fortsetzung):
    - Die *n*-te Gerade alleine teilt R in zwei Regionen
    - Wäre die *n*-te Gerade nicht vorhanden, so würde die *(n+1)*-Gerade *R* ebenfalls in zwei Regionen teilen
    - Sind die *n*-te und die *(n+1)*-te Gerade beide vorhanden, so wird *R* jedoch in vier Regionen geteilt
    - Das Vorhandensein der *n*-ten Gerade hat somit zur Folge, dass die *(n+1)*-te Gerade eine weitere Region innerhalb von *R* erzeugt
    - Da sich die *n*-te und die *(n+1)*-te Gerade nur in einem Punkt *p* schneiden, ist *R* die einzige Region, die davon betroffen ist, so dass die *(n+1)*-te Gerade insgesamt *(n+1)* Regionen hinzufügt. ✓
- Zwei Aspekte an diesem Beweis sind interessant:
  - Es wurde eine Eigenschaft über das Wachstum der Anzahl der Regionen bewiesen, anstelle direkt die Anzahl der Regionen zu berechnen
  - Zweimalige Verwendung der Induktionshypothese: für die *n*-te Gerade und für die *(n+1)*-te Gerade unter der Annahme, dass sie die *n*-te Gerade ist

Ein einfaches Färbungsproblem (1)

- Wir betrachten erneut Geraden in einer Ebene, dieses Mal nicht notwendigerweise in allgemeiner Position
- Dieses Mal sollen die entstehenden Flächen so gefärbt werden, dass keine zwei Nachbarn die gleiche Farbe haben („gültige Färbung“):
  - Hier bei werden zwei Flächen als Nachbarn bezeichnet, wenn sie eine gemeinsame Kante haben (also nicht nur einen Punkt)
  - Im nebenstehenden Beispiel gelten folgende Nachbarschaftsbeziehungen: (a, b), (b, c), (c, d), (d, a)
  - Keine Nachbarn sind: (a, c), (b, d)



- Frage: Wieviele Farben werden für eine derartige Färbung benötigt?
  - Das allgemeine Problem, Regionen in einer Ebene so zu färben, dass benachbarte Regionen unterschiedliche Farben haben, hat Mathematiker ungefähr 100 Jahre beschäftigt, und es werden vier Farben benötigt.
  - Werden die Flächen wie hier betrachtet durch (unendlich lange) Geraden gebildet, ist das Problem einfacher.

Ein einfaches Färbungsproblem (2)

- Behauptung: Es ist möglich, die Regionen, die durch *n* Geraden in einer Ebene gebildet werden, mit 2 Farben gültig zu färben.
- Beweis:

  Induktionsanfang: $n = 1$: trivial

  Induktionshypothese: Es ist möglich, die durch < n Geraden gebildeten Regionen mit weniger als oder gleich zwei Farben gültig zu färben

  Induktionsschritt: $< n \rightsquigarrow n$:
  - Dank Induktionshypothese ist lediglich die Frage zu lösen, wie die Färbung geändert werden muss, wenn die n-te Gerade hinzukommt.
  - Wir teilen die Regionen in zwei Bereiche: der eine enthält alle Regionen, die auf der einen Seite der hinzugekommenen Gerade liegen, und der andere enthält alle Regionen, die auf der anderen Seite liegen.
  - Wir invertieren nun die Färbung aller Regionen auf der anderen Seite
  - Waren zwei benachbarte Regionen auf der anderen Seite vorher unterschiedlich gefärbt, sind sie es auch anschließend ✓

Ein komplizierteres Summierungsproblem (1)

- Wir betrachten das folgende Zahlendreieck:

$$
\begin{aligned}
1 &= 1 \\
3 + 5 &= 8 \\
7 + 9 + 11 &= 27 \\
13 + 15 + 17 + 19 &= 64 \\
21 + 23 + 25 + 27 + 29 &= 125
\end{aligned}
$$

- Frage: Was ist die Summe in der *i*-ten Zeile?

  Induktionshypothese: Die Summe der Zeile *i* ist i^3

  Induktionsanfang: Die Hypothese ist offensichtlich korrekt für $i \leq 5$

  Induktionsschritt: $i \rightsquigarrow i + 1$:
  - Die i-te Zeile enthält i Zahlen
  - Die Zahlen sind die ungeraden Zahlen in aufsteigender Reihenfolge
  - Wir werden uns auf die Differenz zweier aufeinander folgender Zeilen konzentrieren

Ein komplizierteres Summierungsproblem (2)

Induktionsschritt (Fortsetzung):

- Somit reicht es zu zeigen, dass die Differenz zwischen der Zeile *i* und der Zeile *i + 1* genau $(i+1)^3 - i^3$ beträgt (das ist wahr für $i \leq 4$)
- Betrachten wir zunächst die Differenz der ersten Zahl in Reihe *i* und der ersten Zahl in Reihe *i + 1* : Da in der *i*-ten Zeile *i* ungerade Zahlen stehen, ist diese Differenz genau $2 \cdot i$
- Insgesamt gibt es *i* solcher Differenzen in der Zeile *i + 1*, so dass die Gesamtdifferenz der ersten *i* Zahlen $2 \cdot i^2$ beträgt
- Weiterhin gibt es in Zeile *i + 1* noch die *(i + 1)*-te Zahl. Wenn wir beweisen könnten, dass Ihr Wert $(i^2 + 3 \cdot i + 1)$ ist, wäre die Behauptung bewiesen, da: $(i+1)^3 - i^3 = 3 \cdot i^2 + 3 \cdot i + 1$
- Wir werden damit einen Induktionsbeweis für einen kleinen Hilfssatz führen: Die letzte Zahl in Zeile *i + 1* ist $i^2 + 3 \cdot i + 1$
- Somit können wir das Problem, die Summe der Elemente einer Zeile zu bestimmen, darauf reduzieren, den Wert des letzten Elements zu bestimmen!

Ein komplizierteres Summierungsproblem (3)

Induktionshypothese: Die letzte Zahl der Zeile *i +1* ist $i^2 + 3 \cdot i + 1$

Induktionsanfang: Die Hypothese ist offensichtlich korrekt für *i* = 1

Induktionsschritt: $i \rightsquigarrow i + 1$:

- Es ist wiederum ausreichend, die Differenz der letzten Elemente zu betrachten, da so die Induktionsvoraussetzung eingesetzt werden kann
- Laut Induktionsvoraussetzung ist das letzte Element der Zeile *i* :

$$(i-1)^2 + 3(i-1) + 1 = i^2 - 2i + 1 + 3i - 3 + 1 = i^2 + i - 1$$

- Weiterhin wissen wir, dass zwischen den Zahlen an gleichen Positionen zweier aufeinanderfolgenden Zeilen *i* und *i+1* die Differenz *2i* besteht.
- Da die letzte Zahl der Zeile *i+1* jedoch an Position *i +1* steht, beträgt die Differenz zur letzten Zahl der Zeile *i* somit *2i+2*
- Die letzte Zahl der Zeile i+1 hat somit den Wert:

$$i^2 + i - 1 + 2i + 2 = i^2 + 3i + 1 \quad \checkmark$$

Ein komplizierteres Summierungsproblem (4)

Induktionsschritt (Fortsetzung):

- Aus diesem Resultat folgt direkt unser Zwischenresultat, dass die Differenz zwischen den Summen der Zeilen *i* und *i+1* genau $(i+1)^3 - i^3$ beträgt.
- Hieraus folgt nach der ursprünglichen Induktionsvoraussetzung wiederum direkt, dass die Summe der Zeile i+1 genau $(i+1)^3$ beträgt. ✓

- Bemerkungen:
  - Dieser Beweis zeigt erneut, dass man einen Beweis manchmal besser in mehrere Teile zerlegt, als zu versuchen den gesamten Beweis in einem Schritt durchzuführen.
    - Anstelle direkt zu zeigen, dass die Summe der *i*-ten Zeile i^3 beträgt, wurden zunächst die Differenz zwischen Zeilen *i* und *i+1* und daraufhin der Wert des letzten Zahl einer Zeile betrachtet.
  - Weiterhin illustriert der Beweis die oft angewendete Technik, bei der Beweisführung „rückwärts“ vorzugehen. Das ursprüngliche Problem wurde schrittweise auf einfachere Probleme zurückgeführt.

Eine einfache Ungleichung (1)

- Behauptung: $\forall n \geq 1 : \sum_{i=1}^{n} \frac{1}{2^i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \frac{1}{2^n} < 1$

Induktionsanfang: $n = 1$: $1/2 < 1$ ✓

Induktionsschritt: $n \rightsquigarrow n + 1$

- Die einzige Information, die uns die Induktionshypothese liefert, ist dass die Summe der ersten *n* Terme < 1 ist.
- Wie kann die Summe erweitert werden, dass sie auch den *n+1* Term enthält?
- Vielleicht führt die Addition von $1/2^{n+1}$ ja dazu, dass der Wert ≥ 1 wird?
- Der Trick besteht darin, die Induktion in einer anderen Reihenfolge anzuwenden.

Eine einfache Ungleichung (2)

Induktionsschritt (Fortsetzung):

- Wir betrachten von der Summe $\frac{1}{2}+\frac{1}{4}+\frac{1}{8}\cdots+\frac{1}{2^n}+\frac{1}{2^{n+1}}$

  die *letzten n* Terme:

$$\frac{1}{4}+\frac{1}{8}+\cdots\frac{1}{2^n}+\frac{1}{2^{n+1}}=\frac{1}{2}\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}\cdots+\frac{1}{2^n}\right)<\frac{1}{2}$$

  - Diese ergeben laut Induktionshypothese einen Wert < 1/2
  - Nun können wir auf beiden Seiten 1/2 hinzu addieren und erhalten den gewünschten Ausdruck für *n+1* ✓
- Bemerkungen:
  - Es ist oft nicht nötig, im Induktionsschritt das letzte Element als *(n+1)*-tes Element zu betrachten, sondern man kann gelegentlich ein anderes Element mit bestimmten Eigenschaften als *(n+1)*-tes Element wählen
  - Wenn es in einem Induktionsbeweis nicht weiter geht, sollte man daher versuchen, so flexibel wie möglich zu sein

Ein Problem aus der Graphentheorie (1)

- Ein Graph G=(V, E) besteht aus einer Menge V von Knoten (vertices) und einer Menge E von Kanten (edges)
- Jede Kante in E entspricht einem Paar von unterschiedlichen Knoten
- Ein Graph kann gerichtet oder ungerichtet sein. Bei gerichteten Graphen ist die Reihenfolge (v, w) der Knoten bei Kanten wichtig. Eine Kante (v, w) können wir uns als Pfeil von v nach w vorstellen
- Bei ungerichteten Graphen ist die Reihenfolge der Knoten (v, w) bei Kanten unwichtig, da Kanten in beiden Richtungen „verwendet“ werden können
- In diesem Beispiel betrachten wir gerichtete Graphen
- Ein Pfad $v_1, v_2, ..., v_k$ ist eine Sequenz von Knoten, die durch die Kanten $(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k) \in E$ verbunden sind
- Ein Knoten u heißt erreichbar von v, wenn ein Pfad von v nach u mit Kanten in E existiert

Ein Problem aus der Graphentheorie (2)

- Sei U eine Teilmenge von V. Dann bezeichnet der *von U induzierte Subgraph* den Graphen H = (U, F) mit F $\subseteq$ E, wobei F alle Kanten aus E enthält von denen beiden Knoten in U enthalten sind.
- Eine *unabhängige Menge* S in einem Graph G = (V, E) ist eine Menge von Knoten, so dass keine zwei Knoten in S durch eine gemeinsame Kante aus E verbunden sind.
- Wir wollen nun den folgenden Satz beweisen:

> Sei G = (V, E) ein gerichteter Graph. Dann existiert eine unabhängige Menge S(G) in G, so dass alle Knoten in G von einem Knoten in S(G) mit einem Pfad einer Länge ≤ 2 erreicht werden können.

Induktionsbehauptung: Der Satz ist richtig für alle Graphen mit weniger als n Knoten.

Induktionsanfang: n ≤ 3 : trivial ✓

Ein Problem aus der Graphentheorie (3)

Induktionsschritt:

- Sei v ein beliebiger Knoten in V. Sei N(v) ={v} $\cup$ {w $\in$ V | (v, w) $\in$ E} die *Nachbarschaft* des Knotens v
- Wir betrachten den durch V \ N(v) induzierten Subgraph H, also den Subgraph, der entsteht, wenn alle Knoten der Nachbarschaft von v und die entsprechenden Kanten entfernt werden
- Da H weniger Knoten hat als G können wir die Induktionshypothese auf H anwenden. Sei S(H) die unabhängige Menge, die durch die Induktionshypothese impliziert wird. Zwei Fälle sind zu unterscheiden:
  - S(H) $\cup$ {v} ist unabhängig. Dann können wir S(G) = S(H) $\cup$ {v} wählen, da jeder Knoten in N(v) von v aus mit Distanz 1 erreicht werden kann und alle Knoten aus G, die nicht in N(v) sind, von den Knoten aus S(h) mit einem Pfad der Länge ≤ 2 aufgrund der Induktionshypothese erreicht werden können.
  - S(H) $\cup$ {v} ist nicht unabhängig. Dann muss es einen Knoten w $\in$ S(H) geben, der eine gemeinsame Kante mit v hat. Da w $\in$ H ist, kann w nicht in N(v) sein, die Kante muss daher (w, v) sein.

Ein Problem aus der Graphentheorie (4)

Induktionsschritt(Fortsetzung):

- In diesem Fall kann v von w aus in einem Schritt erreicht werden und alle Nachbarn von v können von w aus somit in zwei Schritten erreicht werden. Wir setzen daher S(G) = S(H). ✓

- Bemerkungen:
  - Die Menge an „Reduktion“ war bei diesem Beweis nicht fest, sondern hängt von der tatsächlichen Gestalt unseres „Problems“ (G) ab.
  - Das „kleinere Problem“ war auch kein beliebiges kleineres Problem, sondern es war wesentlich durch das größere Problem bestimmt.
  - Es wurden gerade soviele Knoten entfernt, dass der Beweis geführt werden konnte.
  - Manchmal ist es schwierig, die richtige Menge an zu entfernenden Elementen zu finden: werden zu viele Knoten entfernt, wird die Induktionshypothese zu schwach; entfernt man zu wenige Knoten, ist die Induktionshypothese zu stark.
  - Es wurde „starke Induktion“ verwendet, da vorausgesetzt wurde, dass die Behauptung für alle Graphen mit *maximal n* Knoten gilt.

Finden von Kanten-disjunkten Pfaden in einem Graph (1)

- Sei G = (V, E) ein ungerichteter Graph. Man nennt G einen *verbundenen* Graphen, wenn jeder Knoten v $\in$ V von jedem anderen Knoten w $\in$ V aus über einen Pfad mit Kanten aus G erreicht werden kann.
- Zwei Pfade in G heißen *kanten-disjunkt*, wenn keine Kante aus E in beiden Pfaden vorkommt
- Der *Grad* eines Knotens v in einem ungerichteten Graphen ist die Anzahl der Kanten aus E in denen v vorkommt
- Sei O die Menge von Knoten in V, die einen ungeraden *Grad* haben. Wir zeigen zunächst, dass die Menge von Knoten in O gerade ist:
  - Wenn wir alle Grade von Knoten aus V aufsummieren, erhalten wir genau zweimal die Anzahl von Kanten in E, da jede Kante hierbei zweimal gezählt wird.
  - Da aber alle Knoten mit geradem Grad eine gerade Anzahl zu dieser Summe beitragen, muss auch eine gerade Anzahl an Knoten mit ungeradem Grad in G existieren.

Finden von Kanten-disjunkten Pfaden in einem Graph (2)

- Satz:

> Sei G = (V, E) ein verbundener ungerichteter Graph und sei O die Menge an Knoten in V, die einen ungeraden Grad haben.
> Dann können die Knoten in O in Paare eingeteilt werden, zwischen denen kanten-diskjunkte Pfade in G gefunden werden können.

- Wir führen den Beweis per Induktion über die Anzahl der Kanten in E

Induktionshypothese: Der Satz ist wahr für alle verbundenen ungerichteten Graphen mit < *m* Kanten

Induktionsanfang: *m* = *1* : trivial ✓

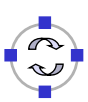
Induktionsschritt:

- Sei G ein Graph mit m Knoten und O die Menge an Knoten mit ungeradem Grad in G.
- Wenn O leer ist, ist der Satz trivialerweise wahr.

Finden von Kanten-disjunkten Pfaden in einem Graph (3)

Induktionsschritt (Fortsetzung):

- Wenn O nicht leer ist, dann wähle zwei beliebige Knoten v, w aus O.
- Da G verbunden ist, existiert ein Pfad von v nach w in G. Entferne alle Kanten auf diesem Pfad aus G. Der resultierende Graph hat weniger als *m* Kanten.
- Wir würden die Induktionshypothese nun gerne anwenden, um Pfade für die restlichen Knoten in O \ {v, w} zu finden.
- Da wir jedoch Kanten aus G entfernt haben, kann es sein, dass der resultierende Graph nicht mehr zusammenhängend ist, und somit kann unsere Induktionshypothese nicht mehr angewendet werden.
- Bei der Reduktion eines Problems muss daher stets darauf geachtet werden, dass nach der Reduktion noch alle Voraussetzungen der Induktionshypothese erfüllt sind!
- In diesem Fall, können wir uns damit behelfen, die Induktionshypothese an unsere Situation anzupassen: Wir werden auf die Bedingung, dass G zusammenhängend sein muss, verzichten.



Finden von Kanten-disjunkten Pfaden in einem Graph (4)

Geänderte Induktionshypothese:

- Der Satz ist wahr für alle ungerichteten Graphen mit < m Kanten.

Induktionsschritt (Fortsetzung):

- Das ist offensichtlich ein stärkerer Satz, der jedoch einfacher zu beweisen ist.
- Wir betrachten wieder einen Graphen mit m Kanten und O der Menge seiner Knoten mit ungeradem Grad. Dieser Graph ist evtl. nicht verbunden, sondern besteht aus einer oder mehrerer sogenannter *Zusammenhangskomponenten* (zwischen denen keine Verbindung existiert).
- Wir wählen nun zwei Knoten v, w aus O, die zur gleichen Komponente gehören. Da jede Komponente wiederum ein zusammenhängender Graph ist, muss sie eine gerade Anzahl von Knoten mit ungeradem Grad haben.
- Wenn wir daher einen Knoten mit ungeradem Grad in einer Komponente finden, muss es auch einen zweiten solchen Knoten geben.

Finden von Kanten-disjunkten Pfaden in einem Graph (5)

Induktionsschritt (Fortsetzung):

- Wir finden nun einen Pfad von v zu w in dieser Komponente und entfernen alle Kanten. Damit erhalten wir einen ungerichteten Graphen mit weniger als m Kanten und können unsere modifizierte Induktionshypothese anwenden.
- In dem verbleibenden Graphen bilden wir nun kanten-disjunkte Pfade zwischen den verbleibenden Knoten mit ungeradem Grad und fügen anschließend unseren zuvor entfernten Pfad wieder hinzu. ✓

- Bemerkungen:
  - Dieser Beweis liefert ein Beispiel für eine mächtige Technik, das “Stärken” der Induktionshypothese.
  - Es erscheint zunächst widersprüchlich, dass ein stärkerer Satz einfacher zu beweisen ist, allerdings haben wir damit auch eine stärkere Induktionshypothese zur Verfügung, die wir zur Lösung unseres “reduzierten” Problems einsetzen können.

Der Arithmetisches vs. Geometrisches Mittel Satz (1)

Satz: Seien $x_1, x_2, \ldots, x_n$ positive Zahlen, dann gilt:

$$(x_1 x_2 \cdots x_n)^{\frac{1}{n}} \leq \frac{x_1 + x_2 + \cdots + x_n}{n}$$

Beweis:

- Der Beweis wird per Induktion über *n* geführt. Die Induktionshypothese ist, dass der Satz wahr ist für *n*.
- Interessant sind an diesem Beweis zwei Aspekte:
  - Wir beweisen den Satz zunächst für alle Zahlen *n,* die eine Zweierpotenz sind, also $n = 2^k$
  - Anschließend verwenden wir das "umgekehrte Induktionsprinzip":

> Wenn ein Satz für eine unendliche Untermenge der natürlichen Zahlen wahr ist, und wenn Ihre Wahrheit für *n* auch ihre Wahrheit für *(n-1)* impliziert, dann ist sie für alle natürlichen Zahlen wahr.

Der Arithmetisches vs. Geometrisches Mittel Satz (2)

- Für unseren Induktionsanfang müssen wir zuerst zeigen, dass der Satz für eine unendliche Menge natürlicher Zahlen gilt
- Wir wählen hierfür die unendliche Menge der Zahlen, die Potenzen von 2 sind: $n = 2^k$

Induktionsanfang:

n = 1: trivial

n = 2: $\sqrt{x_1 x_2} \leq \frac{x_1 + x_2}{2} \Leftrightarrow x_1 x_2 \leq \frac{1}{4}\left(x_1^2 + 2x_1 x_2 + x_2^2\right)$

$$\Leftrightarrow \quad 0 \leq x_1^2 - 2x_1 x_2 + x_2^2$$

$$\Leftrightarrow \quad 0 \leq \left(x_1 - x_2\right)^2 \quad \checkmark$$

Induktionshypothese: Der Satz gilt für n

Der Arithmetisches vs. Geometrisches Mittel Satz (3)

Induktionsschritt: $n \rightsquigarrow 2 \cdot n$

$$(x_1 x_2 \cdots x_{2n})^{\frac{1}{2n}} = \sqrt{(x_1 x_2 \cdots x_n)^{\frac{1}{n}} (x_{n+1} x_{n+2} \cdots x_{2n})^{\frac{1}{n}}}$$

- Wir betrachten die beiden Teilterme y_1 und y_2 mit:

$$y_1 = (x_1 x_2 \cdots x_n)^{1/n} \qquad y_2 = (x_{n+1} x_{n+2} \cdots x_{2n})^{1/n}$$

- Durch Anwenden der Induktionshypothese mit $n = 2$ ergibt sich:

$$(x_1 x_2 \cdots x_{2n})^{\frac{1}{2n}} = \sqrt{y_1 y_2} \leq \frac{y_1 + y_2}{2}$$

- Nochmaliges Anwenden der Induktionshypothese für n ergibt:

$$\frac{y_1 + y_2}{2} \leq \frac{1}{2}\left(\frac{x_1 + x_2 + \cdots + x_n}{n} + \frac{x_{n+1} + x_{n+2} + \cdots x_{2n}}{n}\right)$$ ✓

Der Arithmetisches vs. Geometrisches Mittel Satz (4)

- Wir können nun den Induktionsbeweis für Werte von n führen, die keine Zweierpotenz sind, indem wir aus der Wahrheit des Satzes für n die Wahrheit des Satzes für $n - 1$ folgern:

Induktionsschritt: $n \rightsquigarrow n - 1$

- Wir definieren uns zunächst den Wert z wie folgt: $z = \frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1}$
- Da der Satz für n beliebige positive Zahlen gilt, gilt er insbesondere auch für die n Zahlen $x_1, x_2, \ldots x_{n-1}, z$, also:

$$(x_1 x_2 \cdots x_{n-1} z)^{\frac{1}{n}} \leq \frac{x_1 + x_2 + \cdots + x_{n-1} + z}{n} = \frac{(n-1)z + z}{n} = z$$

- Wir wissen also, dass: $(x_1 x_2 \cdots x_{n-1} z)^{\frac{1}{n}} \leq z$
- Daraus folgt: $x_1 x_2 \cdots x_{n-1} z \leq z^n$
- Und damit: $(x_1 x_2 \cdots x_{n-1})^{\frac{1}{n-1}} \leq z = \frac{x_1 + x_2 + \cdots x_{n-1}}{n-1}$ ✓

Fallstricke bei Induktionsbeweisen (1)

- Wir beweisen per vollständiger Induktion die folgende (unsinnige!) Behauptung:

  Liegen n Geraden so in einer Ebene, dass keine zwei Geraden parallel zueinander verlaufen, dann müssen alle Geraden einen Punkt gemeinsam haben.

  Induktionsanfang: $n = 1$: trivial

  $n = 2$: trivial

  Induktionsschritt: $n \rightsquigarrow n + 1$

  - Gegeben seien $n + 1$ Geraden in der vorausgesetzten Lage, dann treffen sich die ersten n Geraden nach Induktionsvoraussetzung in einem Punkt, ebenso die letzten n Geraden (also ohne die erste aber inkl. der $(n+1)$-ten Gerade)
  - Der gemeinsame Punkt der ersten bzw. letzten n Geraden muss auch allen $n + 1$ Geraden gemeinsam sein, da Geraden, die zwei Punkte gemeinsam haben, auch alle anderen Punkte gemeinsam haben müssen. ✓

Fallstricke bei Induktionsbeweisen (2)

- Wo liegt der Fehler bei diesem Beweis?
  - Damit das Argument des Induktionsschritts greift, muss n mindestens 3 sein, da sich die ersten n Geraden sonst in einem anderen Punkt treffen können als die letzten n Geraden, ohne dass die Geraden zwei Punkte gemeinsam haben müssen.
  - Wie man sich leicht überzeugt, gilt der Satz nicht für $n = 3$
  - Die Behauptung ist richtig für $n = 1$ und $n = 2$
  - Wäre sie auch richtig für $n = 3, 4, ...$ dann würde sie auch für $n = 4, 5, ...$ gelten.
  - Das Problem ist der Schritt von 2 nach 3. Dieses „kleine Problem“ führt dazu, dass der Beweis (und in diesem Fall auch die Behauptung) falsch ist.
- Fallstrick:
  - Der gewählte Induktionsanfang reicht nicht aus, um das Argument des Induktionsschritts anzuwenden

Fallstricke bei Induktionsbeweisen (3)

- Behauptung: $n = \sqrt{1+(n-1)\sqrt{1+n\sqrt{1+(n+1)\sqrt{1+(n+2)\cdots}}}}$

- Damit die Behauptung sinnvoll sein kann, muss der Ausdruck konvergieren. Das ist der Fall (Beweis wird hier nicht angegeben).

  Induktionsanfang: $n = 1$: $1 = \sqrt{1+0(\cdots)}$ ✓

  Induktionsschritt: $n \rightsquigarrow n+1$

  - Da der Satz nach Induktionshypothese für n gilt, können wir einfach beide Seiten der Gleichung quadrieren:

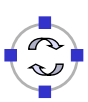
    $$n^2 = 1+(n-1)\sqrt{1+n\sqrt{1+(n+1)\sqrt{1+(n+2)\cdots}}}$$

  - Umformen ergibt:

    $$\frac{n^2-1}{n-1} = n+1 = \sqrt{1+n\sqrt{1+(n+1)\sqrt{1+(n+2)\cdots}}}$$ ✓

Fallstricke bei Induktionsbeweisen (4)

- Der Beweis ist sehr einfach, aber leider falsch. :o(
  - Das Problem liegt darin, dass das Argument nur dann gilt, wenn auf beiden Seiten durch *(n – 1)* dividiert werden kann.
  - Das geht aber nur wenn *(n – 1)* ≠ 0 ist.
  - Unglücklicherweise haben wir als Induktionsanfang *n = 1* gewählt. Das Argument darf in diesem Fall nicht angewendet werden, da die Division durch 0 zu einem undefinierten Ergebnis führt.
- Fallstrick:
  - Das Argument des Induktionsschritts gilt in manchen Fällen nicht.
  - Wenn gerade solche Fälle als „Anker“ der Induktion gewählt werden, „hängt die ganze Induktion in der Luft“
  - Gilt ein Argument in manchen Fällen nicht, entsteht eine Lücke in der Beweiskette. Dieses kann in manchen Fällen durch ein anderes Argument geschlossen werden. Gelingt das nicht, wird der Beweis falsch.
- Bemerkung:
  - Die Behauptung ist übrigens korrekt, ihr Beweis jedoch komplizierter.



Anhang 2: Rekurrenzrelationen

- Eine Rekurrenzrelation (kurz: Rekurrenz) ist eine Methode, eine Funktion durch einen Ausdruck zu definieren, der die zu definierende Funktion selbst enthält
- Beispiel: Fibonacci Zahlen
  - $F(n) = F(n-1) + F(n-2)$ falls $n \geq 2$
  - $F(1) = 1; F(2) = 2$
- Mit dieser Definition kann $F(n)$ für alle natürlichen Zahlen n bestimmt werden, allerdings werden dafür $n-2$ Rechenschritte benötigt
- Frage: Gibt es einen geschlossenen Ausdruck für F?
- Nächste Frage: Warum interessiert uns das in dieser Vorlesung?
  - Rekurrenzrelationen sind ein wichtiges Hilfsmittel bei der Analyse rekursiv formulierter Algorithmen

Abschätzen von Rekurrenzrelationen (1)

- Bei der Aufwandsanalyse von Algorithmen begegnen wir oft Ungleichungen und sind lediglich an einer Abschätzung interessiert
- Beispiel: $T(2n) \leq 2T(n) + 2n - 1; \quad T(2) = 1$
- Können wir eine Funktion $f(n)$ angeben, so dass $T(n) = O(f(n))$ gilt?
  - Diese Schranke sollte möglichst nicht zu unscharf sein.
- Erster Versuch:
  - Wir raten $f(n) = n^2$ und beweisen dies per vollständiger Induktion
  - Induktionsanfang: $T(2) = 1 \leq f(2) = 4$
  - Induktionsschritt: zu zeigen $T(2n) \leq (2n)^2$
    - $T(2n) \leq 2\,T(n) + 2n - 1 \leq 2n^2 + 2n - 1 < (2n)^2$
  - Im letzten Schritt waren wir sehr großzügig, da wir $2n - 1$ durch $2n^2$ abgeschätzt haben
    - Vermutlich ist unsere Schranke daher nicht besonders scharf

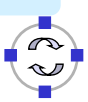
Abschätzen von Rekurrenzrelationen (2)

- Zweiter Versuch: $T(n) \leq c \cdot n$ mit c konstant
  - Da $c \cdot 2 \cdot n = 2 \cdot c \cdot n$ wächst $c \cdot n$ langsamer als $T(n)$, da „kein zusätzlicher Raum“ für die zusätzlichen $2n - 1$ mehr bleibt
  - T(n) wird also irgendwo zwischen c $\cdot n$ und n^2 liegen
- Dritter Versuch: $T(n) \leq n \cdot \log_2 n$
  - Induktionsanfang: $T(2) \leq 2 \cdot \log_2 2$
  - Induktionsschritt: zu zeigen $T(2n) \leq 2 \cdot n \cdot \log_2 2n$
    - $T(2n) \leq 2 \cdot T(n) + 2 \cdot n - 1$
      $\leq 2 \cdot n \cdot \log_2 n + 2 \cdot n - 1$
      $= 2 \cdot n \cdot (\log_2 n + 1) - 1$
      $= 2 \cdot n \cdot \log_2 2n - 1$
      $< 2 \cdot n \cdot \log_2 2n$
    - Bei dieser Abschätzung haben wir im letzten Schritt lediglich die „-1“ weggelassen, so dass die Schranke vermutlich ziemlich scharf ist

Abschätzen von Rekurrenzrelationen (3)

- Bisher ist diese Rekurrenzrelation nur für Zweierpotenzen definiert. Man kann jedoch auch allgemeiner definieren:
  - $T(n) \leq 2\, T(\lfloor n/2 \rfloor) + n - 1$
  - Bei dieser Definition wird mit Hilfe der Gausschen Klammer sichergestellt, dass T nur auf (positive) ganze Zahlen angewendet wird
  - Für alle Zahlen n, die Zweierpotenzen sind, also $n = 2^k$ ist die Rekurrenz äquivalent zu unserem letzten Beispiel
  - Weiterhin ist $T(n)$ monoton wachsend
- Zu zeigen: $T(n) \leq c \cdot n \cdot \log_2 n$
  - Falls n keine Zweierpotenz ist (mit $2^{k-1} \leq n \leq 2^k$), gilt: $T(2^{k-1}) \leq T(n) \leq T(2^k)$
  - Weiterhin wissen wir aufgrund der vorigen Folie, dass für Zweierpotenzen gilt: $T(2^k) \leq c \cdot 2^k \cdot \log_2 2^k$ für eine geeignete Konstante c
  - Beide Ungleichungen zusammen ergeben:
    $T(n) \leq c \cdot 2^k \cdot \log_2 2^k \leq c \cdot 2n \cdot \log_2 2n \leq c_1 \cdot 2^k \cdot \log_2 2^k$
    für eine andere Konstante c_1. Insgesamt: $T(n) = O(n \log n)$

⇒ Es reicht meist aus, Rekurrenzen für Zweierpotenzen abzuschätzen



Abschätzen von Rekurrenzrelationen (4)

Achtung: Beim dem Nachweis einer Abschätzung sollte nicht die O-Notation verwendet werden, da das zu Fehlern führen kann!

Beispiel: $T(2n) \leq 2T(n) + 2n - 1; \quad T(2) = 1$

- Wir raten T(n) = O(n) und ersetzen im Induktionsschritt T(n) durch O(n):
  - $T(2n) \leq 2T(n) + 2n - 1 \leq 2O(n) + 2n - 1 = O(n)$
  - Das ist falsch! (wie wir in unserem zweiten Versuch gesehen haben)

Wenn wir also zeigen wollen, dass $T(n) = O(f(n))$ ist, müssen wir also:

- Ansetzen: $T(n) = c \cdot f(n)$ und
- die Konstante c bestimmen (sofern möglich)

Erst dann können wir schließen: $T(n) = O(f(n))$

Lösung der Fibonacci-Rekurrenz

- $F(n) = F(n-1) + F(n-2); F(1) = 1; F(2) = 2;$
- Raten:
  - Da $F(n)$ die Summe der beiden vorangegangenen Werte ist, könnte man raten, dass sich der Wert von $F(n)$ in jedem Schritt verdoppelt, also 2^n ist:
    - $c \cdot 2^n = c \cdot 2^{n-1} + c \cdot 2^{n-2}$
    - Das ist jedoch unmöglich, da man durch c dividieren kann und die linke Seite immer größer ist als die rechte
  - Nächster Ansatz: $F(n) = a^n$ mit einer anderen Basis a (a konstant):
    - $a^n = a^{n-1} + a^{n-2}$

      $\Rightarrow a^2 = a + 1$
    - Lösen der quadratischen Gleichung ergibt

      $a_1 = (1+\sqrt{5})/2$ und $a_2 = (1-\sqrt{5})/2$
  - Soll eine exakte Lösung bestimmt werden muss allgemein angesetzt werden: $F(n) = c_1 \cdot (a_1)^n + c_2 \cdot (a_2)^n$
  - Das gilt für $c_1 = 1/\sqrt{5}$ und $c_2 = -1/\sqrt{5}$

Asymptotische Abschätzung mit dem Master-Theorem (1)

- Das Master-Theorem hilft bei der Abschätzung von Rekurrenzen der Form: $T(n) = a \cdot T(n/b) + f(n)$
  - Solche Rekurrenzen treten oft bei der Analyse sogenannter Divide-And-Conquer-Algorithmen auf
- Master-Theorem:
  - Seien $a \geq 1$ und $b > 1$ Konstanten. Sei $f(n)$ eine Funktion und sei $T(n)$ über den nicht-negativen ganzen Zahlen durch die folgende Rekursionsgleichung definiert: $T(n) = a \cdot T(n/b) + f(n)$
    - Interpretiere n/b so, dass entweder $\lfloor n/b \rfloor$ oder $\lceil n/b \rceil$
  - Dann kann $T(n)$ wie folgt asymptotisch abgeschätzt werden:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & falls\ gilt: \exists \varepsilon > 0\ mit\ f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \log n) & falls\ gilt: f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & falls: \begin{array}{l} \exists \varepsilon > 0\ mit\ f(n) = \Omega(n^{\log_b a + \varepsilon}) \wedge \\ \exists c < 1 : \exists n_o : \forall n > n_0 : a \cdot f(n/b) \leq c \cdot f(n) \end{array} \end{cases}$$

Asymptotische Abschätzung mit dem Master-Theorem (2)

- Der Beweis ist aufwändiger und wird daher auf fortgeschrittene Vorlesungen verschoben
  - Bei Interesse kann der Beweis im folgenden Buch nachgelesen werden: *Cormen, Leiserson, Rivest & Stein: Algorithmen – eine Einführung. Oldenbourg Verlag, 2004.* (Seiten 75 – 83)
- Hier soll die Anwendung des Theorems an einigen Beispielen vorgeführt werden:
  - $T(n) = 9 \cdot T(n/3) + n$
    - $a = 9; b = 3; f(n) = n$
    - Da $f(n) = O(n^{\log_3 9 - \varepsilon})\ mit\ \varepsilon = 1$ gilt, kann Fall 1 des Master-Theorems angewendet werden
    - Somit gilt: $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$

Asymptotische Abschätzung mit dem Master-Theorem (3)

- Beispiele:
  - *T(n) = T(2n/3) + 1*
    - a = 1; b = 3/2; f(n) = 1
    - Da $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$ ist, gilt $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$ und es kommt Fall 2 des Master-Theorems zur Anwendung
    - Somit gilt $T(n) = \Theta(n^{\log_{3/2} 1} \log n) = \Theta(\log n)$
  - *T(n) = 3 · T(n/4) + n · log n*
    - a = 3; b = 4; f(n) = n · log n
    - Es ist $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$. Somit ist $f(n) = \Omega(n^{\log_4 3 + \varepsilon})$ mit $\varepsilon \approx 0.2$
    - Weiterhin gilt für hinreichend große *n*:
      $a \cdot f(n) = 3 \cdot (n/4) \cdot \log(n/4) \leq (3/4) \cdot n \cdot \log n$
    - Somit kann Fall 3 des Master-Theorems angewendet werden:
      $T(n) = \Theta(f(n)) = \Theta(n \log n)$

Asymptotische Abschätzung mit dem Master-Theorem (4)

- Achtung! Es gibt Fälle, in denen die Struktur der Gleichung „zu passen scheint“, aber kein Fall des Master-Theorems existiert, für den alle Bedingungen erfüllt sind:
  - *T(n) = 2 · T(n/2) + n · log n*
    - *a = 2; b = 2; f(n) = n · log n*
    - Da *f(n) = n · log n* asymptotisch größer ist als $n^{\log_b a} = n$ ist man versucht, Fall 3 des Master-Theorems anzuwenden.
    - Allerdings ist *f(n)* nicht *polynomial größer* als *n*, da für alle positive Konstanten ε > 0 das Verhältnis $f(n) / n^{\log_b a} = n \log n / n = \log n$ asymptotisch kleiner ist als n^{ε}
    - Das Master-Theorem kann in diesem Fall nicht angewendet werden.

Zusammenfassung zu den mathematischen Anhängen

- Vollständige Induktion ist ein mächtiges Verfahren mit dem viele Zusammenhänge über abzählbare Strukturen elegant bewiesen werden können:
  - Wie sich im Verlauf der Vorlesung noch zeigen wird, bestehen starke Korrespondenzen zwischen einem Induktionsbeweis, dass ein Problem eine Lösung hat, und einem Algorithmus, der dieses Problem löst.
  - Oft kann der Algorithmus leicht aus dem Beweis abgelesen werden.
- Rekurrenzrelationen treten häufig bei der Aufwandsanalyse rekursiv formulierter Algorithmen auf:
  - Sie können teilweise durch Raten und Beweisen (per vollständiger Induktion) gelöst bzw. abgeschätzt werden.
  - Bei sogenannten Divide-And-Conquer-Rekurrenzen bietet das Master-Theorem eine elegante Methode für die (scharfe) Abschätzung des asymptotischen Wachstums der durch die Rekurrenz definierten Funktion.
- Ergänzende Literatur:
  - Udi Manber. *Introduction to Algorithms.* Addison-Wesley, 1989.