Diskrete und geometrische Algorithmen

Bernhard Gittenberger

WS 2025/26

1) Einführung

Algorithmen

Algorithmus

- Eingabe (Größe oder Menge von Größen)
- Ausgabe (Größe oder Menge von Größen)
- wohldefinierte Rechenvorschrift

Werkzeug zum Lösen eines Rechenproblems
Formulierung des Problems: gewünschte Eingabe-Ausgabe-Beziehung

Beispiel: Sortierproblem
Eingabe: $(a_1, a_2, \ldots, a_n)$, zB $(27, 15, 34, 79, 55, 25)$
Ausgabe: Permutation $(a'_1, a'_2, \ldots, a'_n)$ von $(a_1, \ldots, a_n)$, sodass $a'_1 \leq a'_2 \leq \cdots \leq a'_n$, also $(15, 25, 27, 34, 55, 79)$

Definition

Eine konkrete Eingabe, die alle geforderten Bedingungen erfüllt, heißt Instanz des Problems.

Algorithmen

Definition

Ein Algorithmus heißt korrekt, wenn er für jede Instanz mit korrekter Ausgabe terminiert.

Spezifikation von Algorithmen

a) natürliche Sprache

b) Computerprogramm (Wahl der Programmiersprache)

c) Pseudo-Code

d) Hardware-Design

Wichtige Probleme

Wichtige Probleme, die algorithmisch behandelt werden

1. Sortierproblem
2. kürzeste Wege
3. lineare Programmierung
4. diskrete Fouriertransformation
5. konvexe Hülle von Punkten
6. maximale Flüsse
7. Zuordnungsproblem
8. Codierung
9. Verschlüsselung

Weitere Aspekte

Datenstrukturen: Methode, um Daten zu speichern und zu organisieren, damit Zugriff und Modifikation leichter werden.

Algorithmen-Design

- Überprüfen der Korrektheit
- Effizienzanalyse (Laufzeit, Speicherbedarf etc.)

Schwere Probleme Probleme, für die keine effizienten Algorithmen bekannt sind.

- P: polynomiell lösbar
- NP: polynomiell verifizierbar (daher $P \subseteq NP$)
- NPC: Klasse von äquivalenten Problemen, für die kein polynomieller Algorithmus bekannt ist.

Analyse von Algorithmen (konkrete Algorithmen) vs. Komplexitätstheorie (Problemklassen)

2) Einführende Beispiele – Sortieralgorithmen

Sortieren durch Einfügen

Eingabe: $(a_1, a_2, \ldots, a_n)$
Ausgabe: Permutation $(a'_1, a'_2, \ldots, a'_n)$ von $(a_1, \ldots, a_n)$, sodass $a'_1 \leq a'_2 \leq \cdots \leq a'_n$
Arbeitsweise: in place, d.h. die Anzahl der a_i, die außerhalb des Arrays gespeichert werden, ist beschränkt.
Notation:

- $A, B, C, \ldots$ Arrays
- $A[i]$: i-ter Eintrag im Array A, $A = [A[1], \ldots, A[n]]$
- $A[i..j]$: Teil-Array, Ausschnitt $[A[i], A[i+1], \ldots, A[j]]$ von A
- $|A|$: Länge des Arrays, $|[A[1], \ldots, A[n]]| = n$

Sortieren durch Einfügen

Algorithm INSERTION-SORT(A)

```
1: for j = 2 to |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:     end while
8:     A[i + 1] := key
9: end for
```

Sortieren durch Einfügen

Algorithm INSERTION-SORT(A)

```
1: for j = 2 to |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:     end while
8:     A[i + 1] := key
9: end for
```

$A =$ (5, 2, 4, 6, 1, 3)

$\rightsquigarrow$ (5, $\boxed{2}$, 4, 6, 1, 3) $\longrightarrow$ (5, $\underline{5}$, 4, 6, 1, 3)

$\rightsquigarrow$ (2, 5, $\boxed{4}$, 6, 1, 3) $\longrightarrow$ (2, 5, $\underline{5}$, 6, 1, 3)

$\rightsquigarrow$ (2, 4, 5, $\boxed{6}$, 1, 3)

$\rightsquigarrow$ (2, 4, 5, 6, $\boxed{1}$, 3) $\longrightarrow$ (2, 4, 5, 6, $\underline{6}$, 3)

$\longrightarrow$ (2, 4, 5, $\underline{5}$, 6, 3)

$\longrightarrow$ (2, 4, $\underline{4}$, 5, 6, 3)

$\longrightarrow$ (2, $\underline{2}$, 4, 5, 6, 3)

$\rightsquigarrow$ (1, 2, 4, 5, 6, $\boxed{3}$) $\longrightarrow \cdots$

$\rightsquigarrow$ (1, 2, 3, 4, 5, 6)$= A$ Ausgabe

Sortieren durch Einfügen

Korrektheit

Verwendung einer Schleifeninvariante

Invariante erfüllt drei Bedingungen:

- Die Aussage ist wahr, bevor die erste Iteration der Schleife beginnt. (Initialisierung)
- Falls die Aussage vor einer Iteration wahr ist, dann auch danach. (Aufrechterhaltung)
- Nach der letzten Iteration beschreibt die Aussage eine nützliche Eigenschaft, die hilft, die Korrektheit zu zeigen. (Terminierung)

Invariante ist zB die Aussage:
„$A[1..j-1]$ besteht aus den ersten $j-1$ Einträgen der Eingabe, jedoch geordnet.“

Sortieren durch Einfügen

Analyse

Vorhersage der Rechenzeit

Annahmen:

- Einprozessormaschine
- RAM Operationen wie
  - Arithmetik ($+, -, *, :,$ mod , $\lfloor\cdot\rfloor, \lceil\cdot\rceil$)
  - Daten verschieben (LOAD, STORE, COPY)
  - Subroutinen zur Steuerung (CALL, RETURN, Verzweigungen, bedingte Verzweigungen)

  haben konstante Laufzeit.
- Für Datentypen wie ganze und Gleitkommazahlen: Länge der Datenwörter beschränkt.
- Konzepte der Speicherhierarchie (Cache, virtueller Speicher) bleiben unberücksichtigt.

Sortieren durch Einfügen

Analyse

Laufzeit abhängig von der Eingabegröße n,
$T(n) =$ Anzahl der Grundoperationen bzw Schritte.

Eingabegröße: abhängig vom Problem,
beim Sortieren: Größe von $(a_1, \ldots, a_n)$ ist n.
Annahme: pro Zeile konstanter Zeitaufwand.

Sortieren durch Einfügen

INSERTION-SORT(A)	Kosten	# Durchläufe
`for` $j = 2$ `to` $\lvert A \rvert$ `do`	c_1	n
$key := A[j]$	c_2	$n-1$
$i := j - 1$	c_3	$n-1$
`while` $i > 0$ `and` $A[i] > key$	c_4	$\sum_{j=2}^{n} t_j$
$A[i+1] := A[i]$	c_5	$\sum_{j=2}^{n}(t_j - 1)$
$i := i - 1$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
`end`	0	
$A[i+1] := key$	c_7	$n-1$
`end do`	0	

$t_j = \#$ Ausführungen der `while`-Schleife für gegebenes j.

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1)$$

Sortieren durch Einfügen

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

Best-Case Analyse: Liste bereits sortiert, d.h. bereits für $i = j - 1$ gilt $A[i] \leq$ key. Daher gilt $t_j = 1$ für $j = 2, \ldots, n$.

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = an + b$$

lineare Laufzeit

Sortieren durch Einfügen

$$\begin{aligned} T(n) =& c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j \\ &+ c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1) \end{aligned}$$

Worst-Case Analyse: $A[j]$ muss mit allen Einträgen von $A[1..j-1]$ verglichen werden, also $t_j = j$ für $j = 2, \ldots, n$. Dann gilt $\sum_{j=2}^{n} t_j = \frac{n(n+1)}{2} - 1$ und $\sum_{j=2}^{n} (t_j - 1) = \frac{n(n-1)}{2} - 1$, also

$$\begin{aligned} T(n) =& \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} \right) n \\ &- (c_2 + c_3 + c_4 + c_7) \\ =& an^2 + bn + c \end{aligned}$$

quadratische Laufzeit

Sortieren durch Einfügen

Average-Case Analyse:

Wahl eines Wahrscheinlichkeitsmodells nötig!
mittlere Laufzeit = Erwartungswert des Modells
Eingabe = Zufallspermutation, d.h. jede Permutation von $(a_1, \ldots, a_n)$ tritt mit Wahrscheinlichkeit $1/n!$ auf.

$$\mathbb{E} = \frac{1}{n!} \sum_{\pi} \text{Laufzeit f. Eingabe } \pi$$

Notation: $\mathrm{rg}(\pi_j) = k : \pi_j$ k.-kleinstes Element von π

Sei $A[1..n]$ zufällige Permutation, dann gilt

$$\mathbb{P}\{\mathrm{rg}(A[j]) = k \text{ in } A[1..j]\} = \frac{1}{j}.$$

Sortieren durch Einfügen

$$\bar{t}_j = \frac{1}{j}(1 + 2 + \cdots + j) = \frac{1}{j}\binom{j+1}{2} = \frac{j+1}{2}$$

$$\sum_{j=2}^{n} \bar{t}_j = \frac{1}{2}\sum_{j=2}^{n}(j+1) = \frac{1}{2}\sum_{j=3}^{n+1} j$$

$$= \frac{1}{2}\left(\frac{(n+1)(n+2)}{2} - 1 - 2\right) = \frac{(n+1)(n+2)}{4} - \frac{3}{2}$$

$$\sum_{j=2}^{n}(\bar{t}_j - 1) = \frac{1}{2}\sum_{j=2}^{n}(j-1) = \frac{1}{2}\sum_{j=1}^{n-1} j = \frac{n(n-1)}{4}$$

$$\bar{T}(n) = \frac{c_5 + c_6 + c_7}{4}n^2 + \left(c_1 + c_2 + c_4 + \frac{3}{4}c_5 - \frac{1}{4}c_6 - \frac{1}{4}c_7 + c_8\right)n$$
$$- (c_2 + c_4 + c_5 + c_8)$$

Sortieren durch Verschmelzen

Idee: Divide and Conquer

- Eingabefolge von n Elementen in zwei etwa gleich große Teilfolgen teilen
- Teilfolgen sortieren (kleinere Instanzen des gleichen Problems)
- Verschmelzen der Teilfolgen

Hilfsprozedur $\textsc{Merge}(A, p, q, r)$

A : Liste; p, q, r : Indizes mit $p \leq q \leq r$;
Voraussetzung: $A[p..q]$ und $A[q+1..r]$ bereits sortiert.

Sortieren durch Verschmelzen

Algorithm MERGE(A, p, q, r)

```
 1: n1 := q − p + 1
 2: n2 := r − q
 3: seien L[1 . . . n1 + 1] und R[1 . . . n2 + 1] zwei neue Felder
 4: for i = 1 to n1 do
 5:     L[i] := A[p + i − 1]
 6: end for
 7: for j = 1 to n2 do
 8:     R[j] := A[q + j]
 9: end for
10: L[n1 + 1] := ∞
11: R[n2 + 1] := ∞
12: i := 1
13: j := 1
14: for k = p to r do
15:     if L[i] ≤ R[j] then
16:         A[k] := L[i]
17:         i := i + 1
18:     else
19:         A[k] := R[j]
20:         j := j + 1
21:     end if
22: end for
```

Korrektheit von MERGE

entscheidender Teil:

```
 1: i := 1
 2: j := 1
 3: for k = p to r do
 4:     if L[i] ≤ R[j] then
 5:         A[k] := L[i]
 6:         i := i + 1
 7:     else
 8:         A[k] := R[j]
 9:         j := j + 1
10:     end if
11: end for
```

Schleifeninvariante: „Vor jeder Iteration in k gilt: (1) $A[p..k-1]$ enthält die $k-p$ kleinsten Einträge von L und R in geordneter Reihenfolge; (2) $L[i]$ und $R[j]$ sind die kleinsten Einträge von L bzw. R, die noch nicht in $A[p..k-1]$ sind.“

Sortieren durch Verschmelzen

„Vor jeder Iteration in k gilt: (1) $A[p..k-1]$ enthält die $k-p$ kleinsten Einträge von L und R in geordneter Reihenfolge; (2) $L[i]$ und $R[j]$ sind die kleinsten Einträge von L bzw. R, die noch nicht in $A[p..k-1]$ sind."

Initialisierung: Wenn $k = p$, dann ist $A[p..p-1]$ leer.

Aufrechterhaltung: Fall 1: $L[i] \leq R[j]$; (Fall 2 analog)
$L[i] \notin A[p..k-1]$, $L[1..i-1] \subseteq A[p..k-1]$;
$A[p..k-1]$ enthält die kleinsten $k-p$ Elemente,
dann $A[k] := L[i]$ und $i := i+1$.

Daher enthält nun $A[p..k]$ die kleinsten $k-p+1$ Elemente und
$L[i] \notin A[p..k]$ sowie $L[1..i-1] \subseteq A[p..k]$.

Terminierung: $k = r+1$;
„$A[p..r]$ enthält die $r+1-p$ $(= n_1+n_2)$ kleinsten Einträge von L und R in geordneter Reihenfolge; $L[n_1+1]$ und $R[n_2+1]$ sind die kleinsten Einträge von L bzw. R, die noch nicht in $A[p..r]$ sind."

Sortieren durch Verschmelzen

Algorithm MERGESORT(A, p, r)

```
1: if p < r then
2:     q = ⌊(p+r)/2⌋                    Divide
3:     MERGESORT(A, p, q)               Conquer
4:     MERGESORT(A, q + 1, r)           Conquer
5:     MERGE(A, p, q, r)                Merge
6: end if
```

Analyse von Divide & Conquer-Algorithmen:

Laufzeit $T(n)$, Ann.: Direkte Lösung für $n \leq c$
Ann.: Divide erzeugt a Teilprobleme der Größe n/b

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n \leq c, \\ aT\left(\frac{n}{b}\right) + D(n) + M(n) & \text{sonst.} \end{cases}$$

Sortieren durch Verschmelzen

Analyse von MERGE-SORT (Worst-Case)

Ann.: $n = 2^k$

$$\left.\begin{array}{rc} \text{Divide:} & D(n) = \Theta(1) \\ \text{Conquer:} & 2T\left(\frac{n}{2}\right) \\ \text{Merge:} & M(n) = \Theta(n) \end{array}\right\} \Longrightarrow T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{für } n > 1. \end{cases}$$

Daraus folgt $T(n) = \Theta(n \log n)$.
Idee: Setze

$$\tilde{T}(n) = \begin{cases} c & \text{falls } n = 1, \\ 2\tilde{T}\left(\frac{n}{2}\right) + cn & \text{für } n > 1. \end{cases}$$

Dann erhält man

$$\tilde{T}(n) = cn + 2\frac{cn}{2} + 4\frac{cn}{4} + \cdots + 2^k\frac{cn}{2^k}.$$

3) Wachstum von Funktionen und elementare Kombinatorik

Asymptotischer Vergleich von Folgen

Asymptotischer Vergleich von Folgen

- $(a_n)_{n\geq 0} \in \mathcal{O}(b_n) :\Leftrightarrow \exists c \in \mathbb{R}$ mit $|a_n| \leq c|b_n|$
- $(a_n)_{n\geq 0} \in o(b_n) :\Leftrightarrow \lim\limits_{n\to\infty} \frac{a_n}{b_n} = 0$
- $(a_n)_{n\geq 0} \in \Theta(b_n) :\Leftrightarrow \exists c_1, c_2 > 0$ mit $c_1|b_n| \leq |a_n| \leq c_2|b_n|$
- $(a_n)_{n\geq 0} \in \Omega(b_n) :\Leftrightarrow b_n = \mathcal{O}(a_n)$
- $(a_n)_{n\geq 0} \in \omega(b_n) :\Leftrightarrow \frac{b_n}{a_n} \to 0$
- $a_n \sim b_n :\Leftrightarrow \lim\limits_{n\to\infty} \frac{a_n}{b_n} = 1$
- $a_n \sim \sum\limits_{m\geq 0} b_{m,n} :\Leftrightarrow \forall M \in \mathbb{N} : a_n \sim \sum\limits_{m=0}^{M} b_{m,n}$,

  $a_n - \sum\limits_{m=0}^{M} b_{m,n} \sim b_{M+1,n}$

Notation: $a_n = \mathcal{O}(b_n)$ für $(a_n)_{n\geq 0} \in \mathcal{O}(b_n)$.

Asymptotischer Vergleich von Folgen

Beispiele

- $n = \mathcal{O}(n^2),\ n^2 + n + 1 = n^2 + \Theta(n)$
-

$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(1 + \frac{1}{12n} + \frac{1}{288n^2} + \sum_{m \geq 3} \frac{c_m}{n^m}\right)$$
$$= \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

Definition

$f(n)$ ist von polynomiellem Wachstum $:\Leftrightarrow \exists k \in \mathbb{N} : f(n) = \mathcal{O}(n^k)$
$f(n)$ ist von polylogarithmischem Wachstum
$:\Leftrightarrow \exists k \in \mathbb{N} : f(n) = \mathcal{O}((\log n)^k)$

Lösen elementarer Rekursionen

Definition

Eine homogene lineare Rekursion k-ter Ordnung mit konstanten Koeffizienten ist eine Rekursion der Form

$$c_k a_{n+k} + c_{k-1} a_{n+k-1} + \cdots + c_1 a_{n+1} + c_0 a_n = 0, \quad n \geq 0$$

mit $c_0, c_1, \ldots, c_k \in \mathbb{R}$, *wobei* $c_0 \neq 0$ *und* $c_k \neq 0$.

Anfangswertproblem:
zusätzlich sind die Werte von $a_0, a_1, \ldots, a_{k-1}$ *vorgegeben.*

Satz

Seien $(a_n^{(1)})_{n\geq 0}$ *und* $(a_n^{(2)})_{n\geq 0}$ *Lösungen der obigen Rekursion, so auch* $(K_1 a_n^{(1)} + K_2 a_n^{(2)})_{n\geq 0}$, *wobei* $K_1, K_2 \in \mathbb{R}$.

Lösen elementarer Rekursionen

Satz

Lösungen $(a_n^{(1)}), (a_n^{(2)}), \ldots, (a_n^{(k)})$ *der Rekursion sind genau dann linear unabhängig, wenn ihre Wronski-Determinante nicht verschwindet, also wenn*

$$\begin{vmatrix} a_0^{(1)} & \ldots & a_0^{(k)} \\ \vdots & & \vdots \\ a_{k-1}^{(1)} & \ldots & a_{k-1}^{(k)} \end{vmatrix} \neq 0.$$

Folgerung

Die Lösungsmenge L der obigen Rekursion ist ein Vektorraum. Die Dimension dieses Vektorraums ist k.

Lösen elementarer Rekursionen

Ansatz: $a_n = \lambda^n$ $(\lambda \neq 0)$

Einsetzen ergibt $c_k\lambda^{n+k} + c_{k-1}\lambda^{n+k-1} + \cdots + c_1\lambda^{n+1} + c_0\lambda^n = 0$

Definition

$\chi(\lambda) := c_k\lambda^k + c_{k-1}\lambda^{k-1} + \cdots + c_1\lambda + c_0$ *heißt charakteristisches Polynom der Rekursion, die Gleichung* $\chi(\lambda) = 0$ *charakteristische Gleichung der Rekursion.*

Wenn $\lambda_1, \lambda_2, \ldots, \lambda_k$ pw verschiedene Nullstellen von $\chi(\lambda)$, dann gilt

$$\begin{vmatrix} \lambda_1^0 & \lambda_2^0 & \ldots & \lambda_k^0 \\ \lambda_1 & \lambda_2 & \ldots & \lambda_k \\ \vdots & \vdots & & \vdots \\ \lambda_1^{k-1} & \lambda_2^{k-1} & \ldots & \lambda_k^{k-1} \end{vmatrix} = \prod_{i<j}(\lambda_j - \lambda_i) \neq 0$$

und somit ist $((\lambda_1^n)_{n\geq 0}, (\lambda_2^n)_{n\geq 0}, \ldots, (\lambda_k^n)_{n\geq 0})$ eine Basis des Lösungsraums.

Lösen elementarer Rekursionen

λ sei ℓ-fache Nullstelle von $\chi(\lambda)$, dann gilt für
$P_{\ell-1}(x) := x^{\ell-1} \frac{\mathrm{d}^{\ell-1}}{\mathrm{d}x^{\ell-1}} (x^n \chi(x))$

$$P_{\ell-1}(\lambda) = (n+k)_{\ell-1} \lambda^{n+k} c_k + (n+k-1)_{\ell-1} \lambda^{n+k-1} c_{k-1} + \cdots + (n)_{\ell-1} \lambda^n c_0 = 0,$$

wobei $(n)_i := n(n-1)\cdots(n-i+1)$.
Analog gilt $P_j(\lambda) = 0$ für $j = 0, \ldots, \ell-2$.
Also sind $(\lambda^n)_{n\geq 0}, (n\lambda^n)_{n\geq 0}, \ldots, (n^{\ell-1}\lambda^n)_{n\geq 0}$ (linear unabhängige) Lösungen der Rekursion.

Satz

Seien $\lambda_1, \lambda_2, \ldots, \lambda_r$ Nullstellen von $\chi(\lambda)$ mit Vielfachheiten $\mu_1, \ldots, \mu_r$, dann lautet die allgemeine Lösung der Rekursion

$$a_n = P_{\mu_1-1}(n)\lambda_1^n + P_{\mu_2-1}(n)\lambda_2^n + \cdots + P_{\mu_r-1}(n)\lambda_r^n,$$

wobei $P_{\mu_1-1}(x), \ldots, P_{\mu_r-1}(x)$ Polynome mit den Graden $\mu_1 - 1, \ldots, \mu_r - 1$ und unbestimmte Koeffizienten sind.

Lösen elementarer Rekursionen

Beispiel

$F_n = F_{n-1} + F_{n-2},\ n \geq 2; \quad F_0 = 0,\ F_1 = 1.$ (Fibonacci-Folge)

$\chi(\lambda) = \lambda^2 - \lambda - 1 = 0$, daher $\lambda_{1,2} = \frac{1}{2} \pm \frac{\sqrt{5}}{2}$

Daraus folgt $F_n = C_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + C_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$

$$F_0 = C_1 + C_2 = 0$$

$$F_1 = C_1 \frac{1+\sqrt{5}}{2} + C_2 \frac{1-\sqrt{5}}{2} = 1$$

Lösen ergibt $C_1 = \frac{1}{\sqrt{5}}$, $C_2 = -\frac{1}{\sqrt{5}}$ und daher

$$F_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Lösen elementarer Rekursionen

Inhomogene lineare Rekursionen mit konstanten Koeffizienten

Satz

Sei $(a_n^{(p)})_{n\geq 0}$ *Lösung der inhomogenen linearen Rekursion*

$$c_k a_{n+k} + c_{k-1} a_{n+k-1} + \cdots + c_1 a_{n+1} + c_0 a_n = s_n, \quad n \geq 0,$$

mit $c_0, c_1, \ldots, c_k \in \mathbb{R}$, *wobei* $c_0 \neq 0$ *und* $c_k \neq 0$, *und* L *die Lösungsmenge der Rekursion*

$$c_k a_{n+k} + c_{k-1} a_{n+k-1} + \cdots + c_1 a_{n+1} + c_0 a_n = 0, \quad n \geq 0.$$

Dann ist

$$(a_n^{(p)})_{n\geq 0} + L$$

die Lösungsmenge der inhomogenen Rekursion.

Lösen elementarer Rekursionen

Beweis: Sei $(a_n)_{n\geq 0} \in M$. Dann gilt

$$\begin{array}{rcccrl}
a_{n+k}c_k & + & \cdots & + & a_n c_0 & = s_n \\
a^{(p)}_{n+k}c_k & + & \cdots & + & a^{(p)}_n c_0 & = s_n \\
\hline
(a_{n+k} - a^{(p)}_{n+k})c_k & + & \cdots & + & (a_n - a^{(p)}_n)c_0 & = 0
\end{array}$$

Daraus folgt $(a_n - a^{(p)}_n)_{n\geq 0} \in L$ und somit $(a_n)_{n\geq 0} \in (a^{(p)}_n)_{n\geq 0} + L$.

Sei nun $(a_n)_{n\geq 0} \in (a^{(p)}_n)_{n\geq 0} + L$.
Dann ist $a_n = a^{(p)}_n + b_n$ mit $(b_n)_{n\geq 0} \in L$ und es gilt

$$\begin{array}{rcccrl}
a^{(p)}_{n+k}c_k & + & \cdots & + & a^{(p)}_n c_0 & = s_n \\
b_{n+k}c_k & + & \cdots & + & b_n c_0 & = 0 \\
\hline
a_{n+k}c_k & + & \cdots & + & a_n c_0 & = s_n
\end{array}$$

Daraus folgt $(a_n)_{n\geq 0} \in M$. □

Spezielle Störfunktionen – Ansatzmethode

Satz

Gesucht ist eine partikuläre Lösung von

$$c_k a_{n+k} + c_{k-1} a_{n+k-1} + \cdots + c_1 a_{n+1} + c_0 a_n = s_n, \quad n \geq 0,$$

wobei $s_n = P_j(n)\alpha^n$ *(*$P_j(x)$ *sei ein Polynom vom Grad* j*).*
Dann führt folgender Ansatz zum Ziel:

$$a_n^{(p)} = q_j(n)\alpha^n n^{\nu(\alpha)},$$

wobei $q_j(x)$ *ein Polynom vom Grad* j *mit unbestimmten Koeffizienten und* α *eine Nullstelle der Vielfachheit* $\nu(\alpha)$ *von* $\chi(\lambda)$ *ist.*

Lösen elementarer Rekursionen

Das Superpositionsprinzip

Satz

Wenn $(\alpha_n)_{n\geq 0}$ *eine partikuläre Lösung von*

$$c_k a_{n+k} + c_{k-1} a_{n+k-1} + \cdots + c_1 a_{n+1} + c_0 a_n = s_n, \quad n \geq 0,$$

ist und $(\beta_n)_{n\geq 0}$ *eine partikuläre Lösung von*

$$c_k a_{n+k} + c_{k-1} a_{n+k-1} + \cdots + c_1 a_{n+1} + c_0 a_n = t_n, \quad n \geq 0,$$

dann ist $(\alpha_n + \beta_n)_{n\geq 0}$ *eine partikuläre Lösung von*

$$c_k a_{n+k} + c_{k-1} a_{n+k-1} + \cdots + c_1 a_{n+1} + c_0 a_n = s_n + t_n, \quad n \geq 0.$$

Lösen elementarer Rekursionen

Beweis: Für $(\alpha_n)_{n\geq 0}$ bzw. $(\beta_n)_{n\geq 0}$ gilt

$$\begin{array}{rcccrcl}
\alpha_{n+k}c_k & + & \cdots & + & \alpha_n c_0 & = & s_n \\
\beta_{n+k}c_k & + & \cdots & + & \beta_n c_0 & = & t_n \\
\hline
(\alpha_{n+k}+\beta_{n+k})c_k & + & \cdots & + & (\alpha_n+\beta_n)c_0 & = & s_n + t_n
\end{array}$$

□

Beispiel

$a_n - a_{n-1} - 8a_{n-2} + 12a_{n-3} = n4^n + 2^n$

$\chi(\lambda) = \lambda^3 - \lambda^2 - 8\lambda + 12 = (\lambda-2)^2(\lambda+3)$

Lösung der homogenen Rekursion $a_n - a_{n-1} - 8a_{n-2} + 12a_{n-3} = 0$:
$a_n^{(h)} = K_1 \cdot 2^n + K_2 \cdot n2^n + K_3 \cdot (-3)^n$

Ansatz für $n4^n$: $a_n^{(1)} = (An+B)4^n$

Ansatz für 2^n: $a_n^{(2)} = n^2 2^n$

Lösung: $a_n = K_1 \cdot 2^n + K_2 \cdot n2^n + K_3 \cdot (-3)^n + a_n^{(1)} + a_n^{(2)}$

Lösen elementarer Rekursionen

Lineare Rekursionen erster Ordnung

homogen: $a_{n+1} = \alpha_n a_n,\ n \geq 0$

Lösung durch Iteration: $a_n = C \prod_{j=0}^{n-1} \alpha_j$

inhomogen: $a_{n+1} = \alpha_n a_n + \beta_n,\ n \geq 0.$

Lösungsstruktur:

$$a_n = a_n^{(h)} + a_n^{(p)}$$

Finden einer partikulären Lösung durch Variation der Konstanten:
Ansatz: $a_n^{(p)} = C_n \prod_{j=0}^{n-1} \alpha_j$

Lösen elementarer Rekursionen

$a_{n+1} = \alpha_n a_n + \beta_n,\ n \geq 0.$
Ansatz: $a_n^{(p)} = C_n \prod_{j=0}^{n-1} \alpha_j$

$$C_{n+1} \prod_{j=0}^{n} \alpha_j = \alpha_n C_n \prod_{j=0}^{n-1} \alpha_j + \beta_n$$

$$C_{n+1} = C_n + \frac{\beta_n}{\prod_{j=0}^{n} \alpha_j}$$

$$C_n = \sum_{\ell=0}^{n-1} \frac{\beta_\ell}{\prod_{j=0}^{\ell} \alpha_j} + C_0$$

Allgemeine Lösung:

$$a_n = \left(\prod_{j=0}^{n-1} \alpha_j \right) \cdot \sum_{\ell=0}^{n-1} \frac{\beta_\ell}{\prod_{j=0}^{\ell} \alpha_j} + C \prod_{j=0}^{n-1} \alpha_j$$