Theoretische Informatik HS23

Nicolas Wehrli

Übungsstunde 02
5. Oktober 2023

ETH Zürich
nwehrl@ethz.ch

Heute

1. Letzte Woche & Feedback zur Serie
2. Kolmogorov Komplexität - Theorie
3. How To Kolmogorov
4. Endliche Automaten - Einführung

Letzte Woche & Feedback zur Serie

Feedback

I. Grundsätzlich gut. Keine Gruppe unter 20 Punkte!
II. Bei manchen die Länge reduzieren und konkreter argumentieren.
III. Bei anderen hat Argumentation gefehlt. Steht in der Korrektur.
IV. 1.b (Fibonacci Aufgabe): Case Distinction auf die letzte Ziffer nicht die vorletzte.
V. Bei Aufgabe 3 musste k und l beliebig bleiben.

Optimierungsprobleme

Ein **Optimierungsproblem** ist ein 6-Tupel $\mathcal{U} = (\Sigma_I, \Sigma_O, L, M, \text{cost}, \text{goal})$, wobei:

(i) Σ_I ist ein Alphabet (genannt **Eingabealphabet**),

(ii) Σ_O ist ein Alphabet (genannt **Ausgabealphabet**),

(iii) $L \subseteq \Sigma_I^*$ ist die Sprache der **zulässigen Eingaben** (als Eingaben kommen nur Wörter in Frage, die eine sinnvolle Bedeutung haben). Ein $x \in L$ wird ein **Problemfall (Instanz) von** $\mathcal{U}$ genannt.

(iv) M ist eine Funktion von L nach $\mathcal{P}(\Sigma_O^*)$, und für jedes $x \in L$ ist $M(x)$ die **Menge der zulässigen Lösungen für** x,

(v) **cost** ist eine Funktion, **cost**: $\bigcup_{x \in L}(\mathcal{M}(x) \times \{x\}) \to \mathbb{R}^+$, genannt **Kostenfunktion**,

(vi) **goal** $\in \{\text{Minimum}, \text{Maximum}\}$ ist das **Optimierungsziel**.

Optimierungsprobleme

Eine zulässige Lösung $\alpha \in \mathcal{M}(x)$ heisst **optimal** für den Problemfall x des Optimierungsproblems $\mathcal{U}$, falls

$$\text{cost}(\alpha, x) = \mathbf{Opt}_{\mathcal{U}}(x) = \text{goal}\{\text{cost}(\beta, x) \mid \beta\mathcal{M}(x)\}.$$

Ein Algorithmus A **löst** $\mathcal{U}$, falls für jedes $x \in L$

(i) $A(x) \in \mathcal{M}(x)$

(ii) $\text{cost}(A(x), x) = \text{goal}\{\text{cost}(\beta, x) \mid \beta \in \mathcal{M}(x)\}.$

Kolmogorov Komplexität - Theorie

Algorithmen generieren Wörter

Sei Σ ein Alphabet und $x \in \Sigma^*$. Wir sagen, dass ein Algorithmus A das Wort x **generiert**, falls A für die Eingabe λ die Ausgabe x liefert.

Beispiel:

```
A_n:    begin
            for i = 1 to n;
                write(01);
        end
```

A_n generiert $(01)^n$.

Aufzählungsalgorithmus

Sei Σ ein Alphabet und sei $L \subseteq \Sigma^*$. A ist ein **Aufzählungsalgorithmus für** L, falls A für jede Eingabe $n \in \mathbb{N} \setminus \{0\}$ die Wortfolge $x_1, ..., x_n$ ausgibt, wobei $x_1, ..., x_n$ die kanonisch n ersten Wörter in L sind.

Aufgabe 2.21

Beweisen Sie, dass eine Sprache L genau dann rekursiv ist, wenn ein Aufzählungsalgorithmus für L existiert.

Das **Entscheidungsproblem** (Σ, L) für ein gegebenes Alphabet Σ und eine gegebene Sprache $L \subseteq \Sigma^*$ ist, für jedes $x \in \Sigma^*$ zu entscheiden, ob

$$x \in L \text{ oder } x \notin L.$$

Ein Algorithmus A **löst** das Entscheidungsproblem (Σ, L), falls für alle $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1, & \text{falls } x \in L, \\ 0, & \text{falls } x \notin L. \end{cases}$$

Wir sagen auch, dass A die Sprache L erkennt.

Aufgabe 2.21

L rekursiv ($\Longrightarrow$) existiert Aufzählungsalgorithmus:

Sei A ein Algorithmus, der L erkennt. Wir beschreiben nun einen Aufzählungsalgorithmus B konstruktiv.

Algorithm 1 $B(\Sigma, n)$

```
i ← 0
while i ≤ n do
    w ← kanonisch nächstes Wort über Σ*
    if A(w) = 1 then
        print(w)
        i ← i + 1
    end if
end while
```

Aufgabe 2.21

Aufzählungsalgorithmus $B \implies L$ rekursiv:

Algorithm 2 $A(\Sigma, w)$

$n \leftarrow |\Sigma|^{|w|+1}$
$L \leftarrow B(\Sigma, n)$
if $w \in L$ **then**
    print(1)
else
    print(0)
end if

Es gibt ein kleines Problem. B könnte unendlich lange laufen, falls $n > |L|$.
Es sollte nicht so schwierig sein, B zu modifizieren, dass es die Berechnung aufhört, falls es keine weiteren Wörter in L gibt.

Information messen

Wir beschränken uns auf Σ_{bool}

Kolmogorov-Komplexität

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ ist die **Kolmogorov-Komplexität** $K(x)$ **des Wortes** x das Minimum der binären Längen, der Pascal-Programme, die x generieren.

$K(x)$ ist die kürzestmögliche Länge einer Beschreibung von x.

Die einfachste (und triviale) Beschreibung von x, ist wenn man x direkt angibt.

x kann aber eine Struktur oder Regelmässigkeit haben, die eine Komprimierung erlaubt.

Welche Programmiersprache gewählt wird verändert die Kolmogorov-Komplexität nur um eine Konstante. (Satz 2.1)

Kolmogorov-Komplexität - Beispiel

Beispiel

Sei $w = 0101010101010101010101010101010101010101$. Die Länge von w ist $|w| = 40$ und die triviale Beschreibunglänge wäre wie gegeben 40.

Aber durch die Regelmässigkeit von einer 20-fachen Wiederholung der Sequenz 01, können w auch durch $(01)^{20}$ beschreiben. Hierbei ist die Beschreibungslänge ein wenig mehr als 4 Zeichen.

Grundlegende Resultate

Es existiert eine Konstante d, so dass für jedes $x \in (\Sigma_{\text{bool}})^*$

$$K(x) \leq |x| + d$$

Die **Kolmogorov-Komplexität einer natürlichen Zahl** n ist $K(n) = K(\text{Bin}(n))$.

Lemma 2.5 - Nichtkomprimierbar

Für jede Zahl $n \in \mathbb{N} \setminus \{0\}$ existiert ein Wort $w_n \in (\Sigma_{\text{bool}})^n$, so dass

$$K(w_n) \geq |w_n| = n$$

Lemma 2.5 - Beweis

Es gibt 2^n Wörter $x_1, ..., x_{2^n}$ über Σ_{bool} der Länge n. Wir bezeichnen $C(x_i)$ als den Bitstring des kürzesten Programms, der x_i generieren kann. Es ist klar, dass für $i \neq j : C(x_i) \neq C(x_j)$.

Die Anzahl der nichtleeren Bitstrings, i.e. der Wörter der Länge $< n$ über Σ_{bool} ist:

$$\sum_{i=1}^{n-1} 2^i = 2^n - 2 < 2^n$$

Also muss es unter den Wörtern $x_1, ..., x_{2^n}$ mindestens ein Wort x_k mit $K(x_k) \geq n$ geben.

■

Satz 2.1 - Programmiersprachen

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ und jede Programmiersprache A sei $K_A(x)$ die Kolmogorov-Komplexität von x bezüglich der Programmiersprache A.

Seien A und B Programmiersprachen. Es existiert eine Konstante $c_{A,B}$, die nur von A und B abhängt, so dass

$$|K_A(x) - K_B(x)| \leq c_{A,B}$$

für alle $x \in (\Sigma_{\text{bool}})^*$.

Ein zufälliges Wort

Ein Wort $x \in (\Sigma_{\text{bool}})^*$ heisst **zufällig**, falls $K(x) \geq |x|$.
Eine Zahl n heisst **zufällig**, falls $K(n) = K(\text{Bin}(n)) \geq \lceil \log_2(n+1) \rceil - 1$.

Jede Binär-Darstellung beginnt immer mit einer 1, deshalb können wir die Länge der Binär-Darstellung um 1 verkürzen.

Zufälligkeit hier bedeutet, dass ein Wort völlig unstrukturiert ist und sich nicht komprimieren lässt. Es hat nichts mit Wahrscheinlichkeit zu tun.

Satz 2.2

Sei L eine Sprache über Σ_{bool}. Sei für jedes $n \in \mathbb{N} \setminus \{0\}$, z_n das n-te Wort in L bezüglich der kanonischen Ordnung. Wenn ein Programm A_L existiert, das das Entscheidungsproblem $(\Sigma_{\text{bool}}, L)$ löst, dann gilt für alle $n \in \mathbb{N} \setminus \{0\}$, dass

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

wobei c eine von n unabhängige Konstante ist.

Satz 2.2 - Beweisidee

Wir können aus A_L, ein Programm entwerfen, dass das kanonisch n-te Wort generiert, indem wir in der kanonischen Reihenfolge alle Wörter $x \in (\Sigma_{bool})^*$ durchgehen und mit A_L entscheiden, ob $x \in L$. Dann können wir einen Counter c haben und den Prozess abbrechen, wenn der Counter $c = n$ wird und dann dieses Wort ausgeben.

Wir sehen, dass dieses Programm ausser der Eingabe n immer gleich ist. Sei die Länge dieses Programms c, dann können wir für das n-te Wort der Sprache L, z_n, die Kolmogorov-Komplexität auf n reduzieren, bzw:

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

Primzahlsatz

Für jede positive ganz Zahl n sei $\mathrm{Prim}(n)$ die Anzahl der Primzahlen kleiner gleich n.

$$\lim_{n\to\infty} \frac{\mathrm{Prim}(n)}{n/\ln n} = 1$$

Nützliche Ungleichung

$$\ln n - \frac{3}{2} < \frac{n}{\mathrm{Prim(n)}} < \ln n - \frac{1}{2}$$

für alle $n \geq 67$.

Lemma 2.6 - schwache Version des Primzahlsatzes

Sei $n_1, n_2, n_3, ...$ eine steigende unendliche Folge natürlicher Zahlen mit $K(n_i) \geq \lceil \log_2 n_i \rceil / 2$. Für jedes $i \in \mathbb{N} \setminus \{0\}$ sei q_i die grösste Primzahl, die die Zahl n_i teilt. Dann ist die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ unendlich.

Beweis: Wir beweisen diese Aussage per Widerspruch:

Nehmen wir zum Widerspruch an, dass die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ sei endlich.

Sei q_m die grösste Primzahl in Q. Dann können wir jede Zahl n_i eindeutig als

$$n_i = q_1^{r_{i,1}} \cdot q_2^{r_{i,2}} \cdot \dots \cdot q_m^{r_{i,m}}$$

für irgendwelche $r_{i,1}, r_{i,2}, ..., r_{i,m} \in \mathbb{N}$ darstellen. Sei c die binäre Länge eines Programms, dass diese $r_{i,j}$ als Eingaben nimmt und n_i erzeugt (A ist für alle $i \in \mathbb{N}$ bis auf die Eingaben $r_{i,1}, ..., r_{i,m}$ gleich).

Lemma 2.6 - Beweis continued

Dann gilt:

$$K(n_i) \leq c + 8 \cdot (\lceil \log_2(r_{i,1} + 1) \rceil + \lceil \log_2(r_{i,2} + 1) \rceil + ... + \lceil \log_2(r_{i,m} + 1) \rceil)$$

Die multiplikative Konstante 8 kommt daher, dass wir für die Zahlen $r_{i,1}, r_{i,2}, ..., r_{i,m}$ dieselbe Kodierung, wie für den Rest des Programmes verwenden (z.B. ASCII-Kodierung), damit ihre Darstellungen eindeutig voneinander getrennt werden können. Weil $r_{i,j} \leq \log_2 n_i, \forall j \in \{1, ..., m\}$ erhalten wir

$$K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil, \forall i \in \mathbb{N} \setminus \{0\}$$

Lemma 2.6 - Beweis continued 2

Weil m und c Konstanten unabhängig von i sind, kann

$$\lceil \log_2 n_i \rceil / 2 \leq K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

$$\lceil \log_2 n_i \rceil / 2 \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

nur für endlich viele $i \in \mathbb{N} \setminus \{0\}$ gelten.

Dies ist ein Widerspruch!

Folglich ist die Menge Q unendlich.

■

How To Kolmogorov

Aufgabentyp 1

Sei $w_n = (010)^{3^{2n^3}} \in \{0,1\}^*$ für alle $n \in \mathbb{N} \setminus \{0\}$. Gib eine möglichst gute obere Schranke für die Kolmogorov-Komplexität von w_n an, gemessen in der Länge von w_n.

Wir zeigen ein Programm, dass n als Eingabe nimmt und w_n druckt:

```
W_n :   begin
            M := n;
            M := 2 × M × M × M;
            J := 1;
            for I = 1 to M
                J := J × 3;
            for I = 1 to J;
                write(010);
        end
```

Aufgabentyp 1

Der einzige variable Teil dieses Algorithmus ist n. Der restliche Code ist von konstanter Länge. Die binäre Länge dieses Programms kann von oben durch

$$\lceil \log_2(n+1) \rceil + c$$

beschränkt werden, für eine Konstante c.

Somit folgt

$$K(w_n) \leq \log_2(n) + c'$$

Wir berechnen die Länge von w_n als $|w_n| = |010| \cdot 3^{2n^3} = 3^{2n^3+1}$.

Aufgabentyp 1

Mit ein wenig umrechnen erhalten wir

$$n = \sqrt[3]{\frac{\log_3 |w_n| - 1}{2}}$$

und die obere Schranke

$$K(w_n) \leq \log_2 \left(\sqrt[3]{\frac{\log_3 |w_n| - 1}{2}} \right) + c' \leq \log_2 \log_3 |w_n| + c''$$

Aufgabentyp 2

Geben Sie eine unendliche Folge von Wörtern $y_1 < y_2 < ...$ an, so dass eine Konstante $c \in \mathbb{N}$ existiert, so dass für alle $i \geq 1$

$$K(y_i) \leq \log_2 \log_2 \log_3 \log_2(|y_i|) + c$$

Wir definieren die Folge $y_1, y_2, ...$ mit $y_i = 0^{2^{3^{2^i}}}$ für alle $i \in \mathbb{N}$. Da $|y_i| < |y_{i+1}|$ folgt die geforderte Ordnung.

Es gilt

$$i = \log_2 \log_3 \log_2 |y_i| \text{ für } i \geq 1$$

Wir zeigen ein Programm, dass i als Eingabe nimmt und y_i druckt:

Aufgabentyp 2

```
begin
        M := i;
        M := 2^(3^(2^M));
        for I = 1 to M;
                write(010);
end
```

Das ^ für die Exponentiation ist nicht Teil der originalen Pascal Syntax, aber wir verwenden es um unser Programm lesbarer zu machen.

Aufgabentyp 2

Der einzige variable Teil dieses Programms ist das i. Der Rest hat konstante Länge. Demnach kann die Länge diese Programms für eine Konstante c' durch

$$\lceil \log_2(i+1) \rceil + c'$$

von oben beschränkt werden.

Somit folgt

$$\begin{aligned} K(y_i) &\leq \log_2(i) + c \\ &\leq \log_2 \log_2 \log_3 \log_2 |y_i| + c \end{aligned}$$

für eine Konstante c.

Aufgabentyp 3

Sei $M = \{7^i \mid i \in \mathbb{N},\ i \leq 2^n - 1\}$. Beweisen Sie, dass mindestens sieben Achtel der Zahlen in M Kolmogorov-Komplexität von mindestens $n - 3$ haben.

Wir zeigen, dass höchstens $\frac{1}{8}$ der Zahlen $x \in M$ eine Kolmogorov-Komplexität $K(x) \leq n - 4$ haben.

Nehmen wir zum Widerspruch an, dass M mehr als $\frac{1}{8}|M|$ Zahlen x enthält, mit $K(x) \leq n - 4$.

Die Programme, die diese Wörter generieren, müssen paarweise verschieden sein, da die Wörter paarweise verschieden sind.

Es gibt aber höchstens

$$\sum_{k=0}^{n-4} 2^k = 2^{n-3} - 1 < \frac{1}{8}|M|$$

Bitstrings mit Länge $\leq n - 4$. **Widerspruch.**

Endliche Automaten - Einführung

Erster Ansatz zur Modellierung von Algorithmen

Ein (deterministischer) **endlicher Automat (EA)** ist ein Quintupel $M = (Q, \Sigma, \delta, q_0, F)$, wobei

(i) Q eine endliche Menge von **Zuständen** ist,

(ii) Σ ein Alphabet, genannt **Eingabealphabet**, ist,

(iii) $q_0 \in Q$ der Anfangszustand ist,

(iv) $F \subseteq Q$ die **Menge der akzeptierenden Zustände** ist und

(v) $\delta : Q \times \Sigma \to Q$ die **Übergangsfunktion** ist.

Konfigurationen

Eine **Konfiguration** von M ist ein Tupel $(q, w) \in Q \times \Sigma^*$.

- "M befindet sich in einer Konfiguration $(q, w) \in Q \times \Sigma^*$, wenn M im Zustand q ist und noch das Suffix w eines Eingabewortes lesen soll."
- Die Konfiguration $(q_0, x) \in \{q_0\} \times \Sigma^*$ heisst die **Startkonfiguration von M auf x**.
- Jede Konfiguration aus $Q \times \{\lambda\}$ nennt man **Endkonfiguration**.

Ein **Schritt** von M ist eine Relation (auf Konfigurationen) $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$, definiert durch

$$(q, w) \vdash_M (p, x) \iff w = ax, a \in \Sigma \text{ und } \delta(q, a) = p.$$

Berechnungen

Eine **Berechnung** C von M ist eine endliche Folge $C = C_0, C_1, ..., C_n$ von Konfigurationen, so dass

$$C_i \vdash_M C_{i+1} \text{ für alle } 0 \leq i \leq n-1.$$

C ist die **Berechnung von** M **auf einer Eingabe** $x \in \Sigma^*$, falls $C_0 = (q_0, x)$ und $C_n \in Q \times \{\lambda\}$ eine Endkonfiguration ist.

Falls $C_n \in F \times \{\lambda\}$, sagen wir, dass C eine **akzeptierende Berechnung** von M auf x ist, und dass M **das Wort** x **akzeptiert**.

Falls $C_n \in (Q \setminus F) \times \{\lambda\}$, sagen wir, dass C eine **verwerfende Berechnung** von M auf x ist, und dass M **das Wort** x **verwirft (nicht akzeptiert)**.

Transitivität von $\vdash_M$ und δ

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Wir definieren $\vdash_M^*$ als die reflexive und transitive Hülle der Schrittrelation $\vdash_M$ von M; daher ist

$$(q, w) \vdash_M^* (p, u) \iff (q = p \wedge w = u) \text{ oder } \exists k \in \mathbb{N} \setminus \{0\},$$

so dass

(i) $w = a_1 a_2 ... a_k u, a_i \in \Sigma$ für $i = 1, 2, ..., k$, und

(ii) $\exists r_1, r_2, ..., r_{k-1} \in Q$, so dass

$$(q, w) \vdash_M (r_1, a_2 ... a_k u) \vdash_M ... \vdash_M (r_{k-1}, a_k u) \vdash_M (p, u)$$

Transitivität von $\vdash_M$ und δ

Wir definieren $\hat{\delta} : Q \times \Sigma^* \to Q$ durch:

(i) $\hat{\delta}(q, \lambda) = q$ für alle $q \in Q$ und

(ii) $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ für alle $a \in \Sigma, w \in \Sigma^*, q \in Q$.

$$\hat{\delta}(q, w) = p \iff (q, w) \vdash_M^* (p, \lambda)$$

Die **von M akzeptierte Sprache** $L(M)$ ist definiert als

$$\begin{aligned} L(M) &= \{w \in \Sigma^* \mid \text{Berechnung von } M \text{ auf } w \text{ endet in } (p, \lambda) \in F \times \{\lambda\}\} \\ &= \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (p, \lambda) \wedge p \in F\} \\ &= \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\} \end{aligned}$$

$\mathcal{L}_{\mathbf{EA}} = \{L(M) \mid M \text{ ist ein EA}\}$ ist die Klasse der Sprachen, die von endlichen Automaten akzeptiert werden.
$\mathcal{L}_{\text{EA}}$ bezeichnet man auch als die **Klasse der regulären Sprachen**, und jede Sprache $L \in \mathcal{L}_{\text{EA}}$ wird **regulär** genannt.

Klassen für alle Zustände im Endlichen Automaten

Für alle $p \in Q$ definieren wir die Klasse

$$\begin{aligned}\mathbf{Kl[p]} &= \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) = p\} \\ &= \{w \in \Sigma^* \mid (q_0, w) \Big|_M^* (p, \lambda)\}\end{aligned}$$

Wir bemerken dann

$$\bigcup_{q \in Q} \mathrm{Kl[q]} = \Sigma^*$$

$$\mathrm{Kl}[q] \cap \mathrm{Kl}[p] = \emptyset, \forall p, q \in Q, p \neq q$$

$$L(M) = \bigcup_{q \in F} \mathrm{Kl}[q]$$

Typische Aufgabe

Entwerfen sie für folgende Sprache einen Endlichen Automat und geben Sie eine Beschreibung von Kl$[q]$ für jeden Zustand $q \in Q$.

$$L_1 = \{xbbya \in \{a, b\}^* \mid\}$$