PRIMES is in P

Ein Vortrag von Holger Szillat

szillat@informatik.uni-tuebingen.de

Übersicht

- *Geschichte*
- *Notationen und Definitionen*
- *Der Agrawal-Kayal-Saxena-Algorithmus*
- *Korrektheit und Aufwand*
- *Fazit*

Geschichte

- Altes Problem: Ist n eine Primzahl?
- Meistens theoretischer Natur…“p sei eine Primzahl”
- Aber: Konkret berechnen? ($\rightarrow$ PRIMES)
- Computer + Internet $\rightarrow$ Kryptographie

Konkret berechnen

"Probe-Dividieren":

- Probiere *alle* (ungeraden) Zahlen $m \leq \sqrt{n}$, ob sie n teilen.
- Problem: Aufwand etwa $\sqrt{n}$.
- Für grosse n (mit mehreren hundert Stellen) nicht praktikabel.

Konkret berechnen

“Sieb des Eratosthenes” (ca. 240 vor Christus):

- Schreibe alle Zahlen $\leq n$ hintereinander auf.
- Beginne bei 2, streiche alle Vielfachen.
- Nehme nächste nicht gestrichene Zahl, streiche alle Vielfachen.
- $\dots$
- Findet *alle* Primzahlen $p \leq n$.
- Problem: Zuviele Primzahlen und Aufwand $(\frac{n}{2})$!

Konkret berechnen

Ziel:

- Generell: Aufwand sollte nur von der Länge der Eingabe abhängen.
- Hier: “Länge” der Zahl n.
- n ist binär kodiert: $|n| = \lfloor \log_2(n) \rfloor + 1$.
- Beispiele: **Miller/Rabin-** oder **Solovay/Strassen-**Verfahren.

Miller/Rabin-Verfahren

- Eine Zahl n wird als “nicht-prim” deklariert: $\surd$
- Eine Zahl n wird als “prim” deklariert: Vielleicht!
  $\Rightarrow$ n kann auch zusammengesetzt sein.
- Idee: Angenommen n ist prim, dann $(n-1) = 2^s \times d$, mit $s \in \mathbb{N}$ und d ungerade. Dann gilt für ein (zu n teilerfremdes) a entweder:
  $a^d \equiv 1 \pmod{n}$ oder $a^{2^r d} \equiv 1 \pmod{n}$ mit $0 \leq r \leq s-1$.
- Wähle $a \in \mathbb{Z}/n\mathbb{Z}$ zufällig und prüfe.
- (Sollte die *Erweiterte Riemann Hypothese* gelten, dann…)
- Aufwand ist etwa $\mathrm{O}(c \log^3(n))$
- Algorithmus ist *randomisiert* mit $\epsilon \leq \frac{1}{4^k}$.

Solovay/Strassen-Verfahren

- Eine Zahl n wird als "nicht-prim" deklariert: $\surd$
- Eine Zahl n wird als "prim" deklariert: Vielleicht!
  $\Rightarrow$ n kann auch zusammengesetzt sein.
- Idee: Für eine Primzahl p gilt: $\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \pmod{n}$
- Wähle $a \in \mathbb{Z}/n\mathbb{Z}$ zufällig und prüfe.
- Aufwand ist etwa $\mathrm{O}(c \log^3(n))$
- Algorithmus ist *randomisiert* mit $\epsilon \leq \frac{1}{2^k}$

Jacobi-Symbol

Definition (Jacobi Symbol). Sei n eine ungerade Zahl mit der Primfaktorisierung $p_1^{k_1} p_2^{k_2} \dots p_t^{k_t}$, dann ist für alle a mit $\mathrm{ggT}(a, n) = 1$ das *Jacobi Symbol* definiert als:

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{t} \left(\frac{a}{p_i}\right)^{k_i}$$

Probleme

- Algorithmen sind randomisiert, d.h. ein Ergebnis kann falsch sein.
- Sowohl *Miller/Rabin* als auch *Solovay/Strassen* basieren auf unbewiesener Vermutung.
- Aber: Brauchbar!

Allgemeine Probleme

- Primzahlproblem vielleicht gar nicht mit Aufwand “log()” lösbar? (Zumindest nicht randomisiert?!)
- Bekannt: Primzahl-Problem in NP
- $P \subseteq NP$
- Umgekehrt: (Unbekannter) Algorithmus gibt zurück:
  n **NICHT PRIM!**
  Nachweis durch Finden eines Faktors von n möglich.
- Aber: Dauert genau so lange!
- Damit: “Nicht-Primzahl-Problem” auch in NP (co-NP)
  $\Rightarrow$ Primzahl-Problem in $NP \cap co\text{-}NP$

Komplexitätsklassen

Was sind P und NP überhaupt?

- P ist die Menge aller Probleme, für die es einen deterministischen Algorithmus gibt, der eine Lösung in polynomieller Zeit findet.
- NP ist die Menge aller Probleme für die ein nicht-deterministischer Algorithmus in polynomieller Zeit eine Lösung findet. (Gibt's nicht!)

  Beispiel *Teilsummen-Problem*: Gegeben: Eine Menge $M = \{a_1, a_2, \ldots, a_n\}$ mit $a_i \in \mathbb{N}$ und $s \in \mathbb{N}$. Frage: Gibt es eine Teilmenge $T \subseteq M$ mit $s = \sum T$?

Für das *Teilsummen-Problem* ist kein Algorithmus $\in P$ bekannt. Aber: Problem kann $\in P$ verifiziert werden!

Weitere interessante Komplexitätsklassen

- *ZPP (Zero-error probabilistic polynomial time)*: Algorithmus liefert ein korrektes Ergebnis ("Ja"/"Nein") oder liefert mit einer Wahrscheinlichkeit ϵ ein "Unbekannt"
- *BPP (Bounded-error probabilistic polynomial time)*: Algorithmus liefert ein Ergebnis, das mit einer gewissen Wahrscheinlichkeit *epsilon* falsch ist. (sog. *Monte-Carlo-Algorithmen*)
- *RP (Random polynomial time)*: Algorithmus liefert entweder ein deutliches "Nein" oder mit einer Wahrscheinlichkeit *epsilon* ein "Ja".

Beziehungen

- $P \subseteq ZPP \subseteq RP \subseteq NP$
- $P \subseteq ZPP \subseteq co\text{-}RP \subseteq BPP$.

Grosses, offenes Problem in der Komplexitätstheorie:

$$P = NP$$

In welcher Klasse ist ein Algorithmus?

- Schreibe Programme, messe Zeit.
- Probleme:
  1. “Mein Computer ist schneller als Deiner!”
  2. “Meine Programmiersprache ist besser als Deine!”
  3. “Meine Eingabe ist besser als Deine!”
  4. “Aber mein Algorithmus ist doch besser?” (Um wieviel?)
- **Also so nicht!**

Turing-Maschine

Man braucht ein Gerät, welches "unabhängig" ist von Hard- und Software:

- *Turing-Maschine* (Alan Turing, 1936)
- Er wollte "eine mathematisch präzise Definition eines Algorithmus' "
- Kann jedes real existierende Rechnermodell mit nur geringen Rechenzeitverlusten simulieren.

D.h. braucht ein Algorithmus **A**, auf einer Turing-Maschine t Rechenschritte, dann existiert ein Polynom p, so dass eine real existierende Maschine höchstens $p(t)$ Rechenschritte braucht.

Turing-Maschine

Eine Turing-Maschine besteht aus:

1. einem Band, welches in Zellen unterteilt ist: Speicher
2. einem Schreib-/Lesekopf: liest/schreibt Symbole von/auf Band. Bewegt sich immer nur um eine Zelle nach links/rechts.
3. einem Zustands-Register; speichert den aktuellen Zustand der Turing-Maschine.
4. endlich vielen Zustände; Register wird zum Start mit Startzustand initialisiert.
5. einer Aktionstabelle oder Übergangsfunktion: Programm

Turing-Maschine

Formal besteht eine Turing-Maschine aus einem 5-Tupel $M = (Q, \Gamma, s, F, \delta)$ mit:

- Q: Menge der Zustände
- Γ: endliche Menge des Bandalphabets
- $s \in Q$ ist Startzustand
- $F \subseteq Q$: Menge der finalen Zustände
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ die Übergangsfunktion.
  $(q, X) \mapsto (p, Y, D)$ mit $p, q \in Q$; $X, Y \in \Gamma$ und $D \in \{L, R\}$.

  D.h. wird in einem Zustand q das Symbol X gelesen, dann versetze die Turing-Maschine in den Zustand p, schreibe das Symbol Y und bewege den Kopf nach Links oder Rechts.

Beispiel

Beispiel. Gegeben sei eine Menge $A = \{0^{2^n} | n \geq 0\}$, also die Menge, die aus dem Zeichen 0 besteht und deren Zeichenketten die Länge 2^n haben, z.B. $A = \{0, 00, 0000, \ldots\}$.

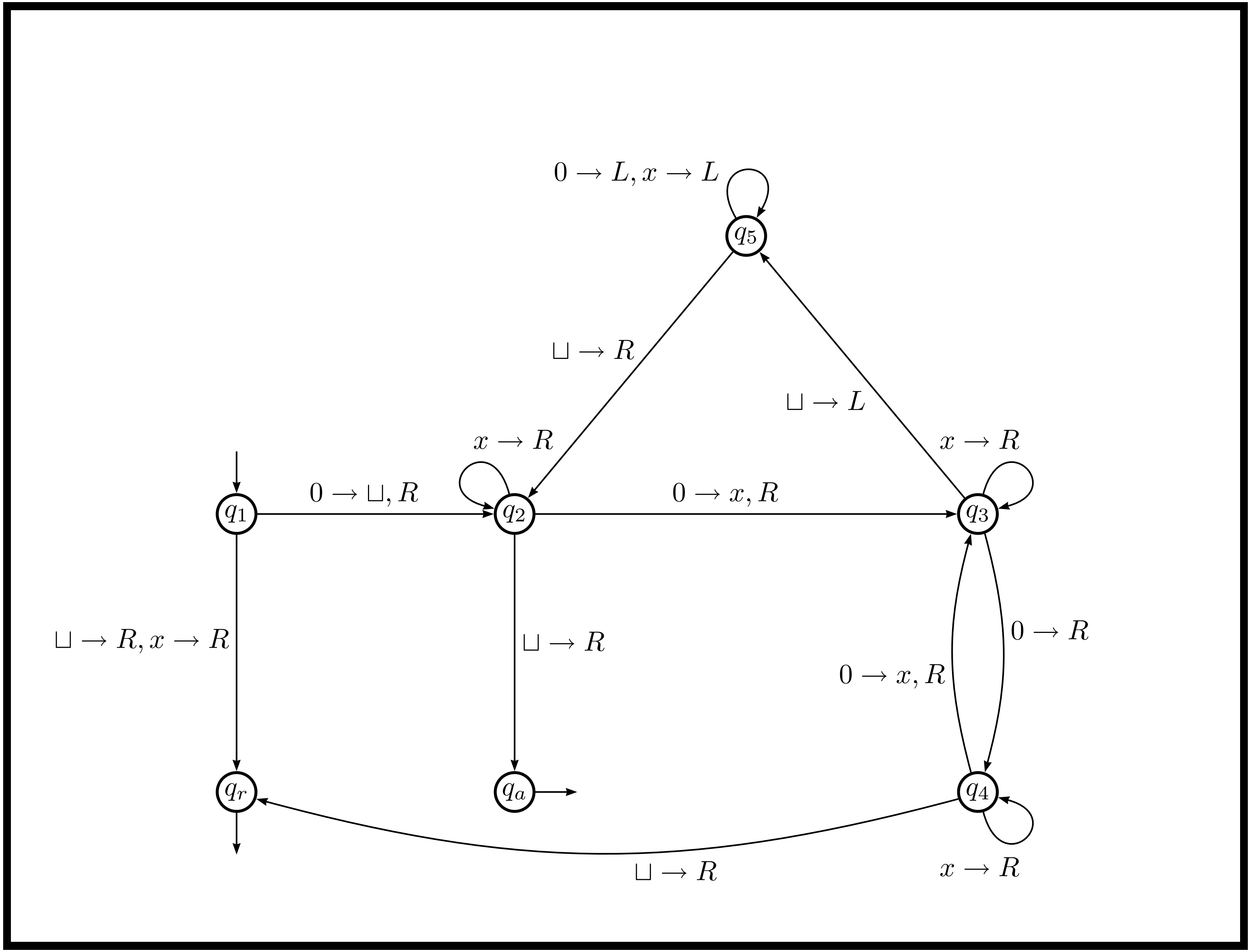
Die Turing-Maschine, welche die Menge (Sprache) A akzeptiert, ist gegeben durch:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\}$
- $\Sigma = \{0\}$
- $\Delta = \{0, x, \sqcup\}$
- $\Gamma = \Sigma \cup \Delta$
- δ sei beschrieben durch Abbildung.
- Der Startzustand sei q_1
- $F = \{q_{\text{accept}}, q_{\text{reject}}\}$

Beispiel

Informell lässt sich der Algorithmus so beschreiben:

1. Bewege den Lesekopf von links nach rechts über das Band, lösche dabei jede zweite 0 und speichere die Anzahl der gelöschten 0.
2. Wenn nur eine 0 gelöscht wurde: “accept”.
3. Wenn mehr als eine 0 gelöscht wurde und die Anzahl der gelöschten 0 ungerade ist: “reject”.
4. Bewege den Lesekopf wieder zurück an den Anfang des Bandes.
5. Gehe in Zustand 1.



$0 \to L, x \to L$
q_5
$\sqcup \to R$
$\sqcup \to L$
$x \to R$
$x \to R$
$0 \to \sqcup, R$
$0 \to x, R$
q_1
q_2
q_3
$\sqcup \to R, x \to R$
$\sqcup \to R$
$0 \to R$
$0 \to x, R$
q_r
q_a
q_4
$\sqcup \to R$
$x \to R$

Turing-Maschine

Meistens nur interessant:

- "Wie lange braucht die Turing-Maschine?"
- "Wird das Problem gelöst oder nicht?": Problem definiert eine Menge L_p an Lösungen, daher: "Ist Eingabe $x \in L_p$?"
- Turing-Maschine repräsentiert L_p
- L_p definiert eine "Sprache"
- Daher: Akzeptiert die Turing-Maschine eine Eingabe x? ("Turing-Maschine akzeptiert Eingabe x" $\Leftrightarrow x \in L_p$)

Wie vergleicht man zwei Algorithmen?

- Turing-Maschine macht unabhängig von Hard- und Software.
- Bleibt noch: “Wieviel ist Algorithmus **A** schneller als **B**?”
- Wie gross ist der Aufwand von Algorithmus **A**?
- Lösung: O-Notation

O-Notation

- Beschreibt eine asymptotische obere Grenze für das Wachstum einer Funktion.
- Beispiel: Funktion $T(n) = 4n^2 - 2n + 2$
- Für wachsendes $n \to \infty$ überwiegt der Term n^2
- Daher: $T \in \mathrm{O}(n^2)$

O-Notation

Definition (O-Notation). Sei $f : \mathbb{N} \to \mathbb{N}$, dann ist $f(x) = \mathrm{O}(g(x))$, gdw. $\exists k \in \mathbb{N} : \exists x_0 \in \mathbb{N} : \forall x \geq x_0 : f(x) \leq kg(x)$.

Bemerkung. Die Schreibweise $f = \mathrm{O}(g(n))$ ist daher eigentlich falsch. Korrekt wäre: $f(x) \in \mathrm{O}(g(x))$, denn: $f(x) = \mathrm{O}(g(x)) \Leftrightarrow \mathrm{O}(g(x)) = f(x)$, was aber nicht zur obigen Definition passt.

Ω-Notation

Analog zur O-Notation für eine "asymptotische obere Grenze", gibt es die Ω-Notation für eine "asymptotische untere Grenze".

Definition. Sei $f : \mathbb{N} \to \mathbb{N}$, dann ist $f(x) = \Omega(g(x))$, gdw. $\exists k \in \mathbb{N} : \exists x_0 \in \mathbb{N} : \forall x \geq x_0 : f(x) \geq kg(x)$.

Θ-Notation

Für manche Algorithmen ist es möglich "asympototisch enge Grenzen" anzugeben.

Definition. Sei $f : \mathbb{N} \to \mathbb{N}$, dann ist
$f(x) \in \Theta(g(x))$ gdw. $f(x) = \mathrm{O}(g(x))$ und $g(x) = \mathrm{O}(f(x))$.

Vielleicht: PRIMES $\notin \boldsymbol{P}$?

- Bisher klar: PRIMES $\in NP \cap co\text{-}NP$
- Bisher unklar: PRIMES $\in P$?
- Dann kamen *Mannindra Agrawal*, *Neeraj Kayal* und *Nitin Saxena* zeigten: PRIMES $\in P$.
- Warum?

Idee: Fermats kleines Theorem

Theorem (Fermats kleines Theorem). *Ist n prim, so gilt für alle $a \in \mathbb{Z}$ mit $ggT(a, n) = 1$:*

$$a^{p-1} \equiv 1 \pmod{p}$$

Damit möglich: Prüfe ob für (alle) a und gegebenes n gilt:

$$n \in \mathbb{P} \Rightarrow a^{n-1} \equiv 1 \pmod{n}$$

Leider auch die *Carmichael-Zahlen*!

Carmichael-Zahlen

Definition (Carmichael-Zahl). Eine (zusammengesetzte) Zahl $n \in \mathbb{N}$ heißt *Carmichael-Zahl*, gdw. für alle $a \in \mathbb{Z}/n\mathbb{Z}$ gilt:

$$a^{n-1} \equiv 1 \pmod{n}$$

Das kleinste Beispiel einer Carmichael-Zahl ist 561, welche in $3 \times 11 \times 17$ faktorisiert werden kann.

Idee des Algorithmus'

Verallgemeinerung von *Fermats kleinem Theorem*:

Lemma. *Sei* $a \in \mathbb{Z}$, $n \in \mathbb{N}$, $n \geq 2$ *und* $ggT(a, n) = 1$. *Dann ist* n *prim, gdw.:*

$$(X + a)^n = X^n + a \pmod{n}$$

Require: $n \in \mathbb{N}, n > 1$

if $(n = a^b, a \in \mathbb{N}, b > 1)$ **then**
    COMPOSITE
end if
Finde kleinstes r mit: $o_r(n) > 4\log^2(n)$
if $(1 < \mathrm{ggT}(a, n) < n$ für ein $a \leq r)$ **then**
    COMPOSITE
end if
if $(n \leq r)$ **then**
    PRIME
end if
for $a \leftarrow 1 \ldots \lfloor 2\sqrt{\phi(r)}\log(n) \rfloor$ **do**
    if $((X + a)^n \neq (X^n + a) \pmod{X^r - 1, n})$ **then**
        COMPOSITE
    end if
end for
PRIME

Korrektheit

Proposition. *n ist prim $\Rightarrow$ Algorithmus liefert "**PRIME**"*

Beweis. Wenn n prim ist...

- dann können Schritte 1-3 und 5-7 nicht "**COMPOSITE**" liefern.
- dann kann die `for`-Schleife nicht "**COMPOSITE**" liefern (wegen Lemma).
- also wird entweder in den Schritten 8-10 oder Schritt 16 "**PRIME**" liefern.

$\square$

Aufwand

Theorem. *Algorithmus hat den Aufwand* $\tilde{O}(\log^{10.5}(n))$.

Beweis.

- Arithmetische Operationen haben Aufwand $\tilde{O}(\log^3(n))$
- Schritt 4: $\tilde{O}(\log^2(n)\log(r))$, $\mathrm{O}(\log^5(n))$-mal: $\tilde{O}(\log^7(n))$ $[r = \mathrm{O}(\log^5(n))]$
- Schritt 5-7: ggT finden: Aufwand $\mathrm{O}(\log(n))$; r-mal: $\mathrm{O}(r\log(n)) = \mathrm{O}(\log^6(n)) = \mathrm{O}(\log(n))$
- Schritt 11-15: $\lfloor 2\sqrt{\phi(r)}\log(n)\rfloor$-mal durchlaufen:
  - Jede Gleichung kann mit Aufwand $\tilde{O}(r\log^2(n))$ geprüft werden.
  - Damit: $\tilde{O}(r\sqrt{\phi(r)}\log^3(n)) = \tilde{O}(r^{\frac{3}{2}}\log^3(n)) = \tilde{O}(\log^{10.5}(n))$.

$\square$

Fazit

- Tatsächlich: PRIMES $\in P$
- Leider: Laufzeit schlechter als **Miller/Rabin** oder **Solovay/Strassen**.
- Nur "theoretische Revolution".