Vorlesungsprotokoll vom 09. Januar 2007

Kuniko Lischke-Yoshida

Teil I: Vorlesungsprotokoll
Teil II: Probabilistische Algorithmen
Teil III: Primzahltests

Teil I: Vorlesungsprotokoll

Probabilistischer Algorithmus

- treffen während der Abarbeitung zufällige Wahl, z.B. 'wähle zufällig $a \in$ M (M endliche Menge)
- verschiedene Läufe führen bei gleichem Input zu unterschiedlichen Ergebnissen

Bei Entscheidungsproblemen: Fehlerwahrscheinlichkeit.
Diese kann sein:

einseitig: Antwort NEIN stets korrekt
Antwort JA möglicherweise falsch
zweiseitig: beide Antworten möglicherweise falsch

Klassische Beispiele:
Primzahl oder nicht? (s. Teil II und Teil III)

Wir betrachten nun ein Beispiel:

Gleichheit für Polynome: PIT

Da $p \equiv q$ gdw $p - q \equiv 0$, reicht es, letztes zu testen.

Problem **PIT**:
geg.: Polynom $p(x_1, \cdots, x_n)$ als bel. algebraischer Ausdruck
Frage: Ist $p(x_1, \cdots, x_n) \equiv 0$ d.h. $p(a_1, \cdots, a_n) = 0$
für alle $a_1, \cdots, a_n \in \mathbb{Q}$?
Für dieses Problem ist kein effizienter deterministischer Algorithmus bekannt.

Satz

Sei $p(x_1, \cdots, x_n) \not\equiv 0$ und $S \subseteq \mathbb{Q}$ endlich. Dann ist die Anzahl der $(a_1, \cdots, a_n) \in S^n$ mit $p(a_1, \cdots, a_n) = 0$ höchstens $nd|S|^{n-1}$, wobei d=maximaler Grad einer Variable in $p(x_1, \cdots, x_n)$

Algorithmus:

input $p(x_1, \cdots, x_n)$
berechne d= max. Grad
wähle S mit $|S| = 2n + 1$ (z.B. $[-nd, +nd]$)
wähle $(a_1, \cdots a_n) \in S^n$ zufällig
falls $p(x_1, \cdots, x_n) \neq 0$ verwerfe
sonst akzeptiere

Ist $p(x_1, \cdots, x_n) \equiv 0$, so wird immer akzeptiert.
sonst

ist Pr[akzeptiert] $= \dfrac{nd|S|^{n-1}}{|S|^n} = \dfrac{nd}{|S|} = \dfrac{nd}{2nd+1} < \dfrac{1}{2}$

Beweis des Satzes: durch Induktion nach n

1) $n = 1$ $nd|S|^{n-1} = d$ (nach dem Fundamentalsatz)
2) $n - 1 \to n$
Sei $p(x_1, \cdots, x_n) = \Sigma_{i=1}^{d} p_i(x_2, \cdots, x_n) x_1^i$
Sei i maximal mit $p_i(x_2, \cdots, x_n) \not\equiv 0$
Für $(a_2, \cdots, a_n) \in S^{n-1}$ gibt es zwei Fälle
1. Fall: $p_i(a_2, \cdots, a_n) \neq 0$
Dann gibt es höchstens d Werte $b \in S$ mit
$p(b, a_2, \cdots, a_n) = 0$
$\Rightarrow$ insgesamt höchstens $d|S|^{n-1}$ solche Nullstellen
2. Fall: $p_i(a_2, \cdots, a_n) = 0$
Dann kann $p(b, a_2, \cdots, a_n) = 0$ für alle $b \in S$ sein.
Dies ist aber nach Ind.annahme für höchstens
$(n-1)d|S|^{n-2}$ der $(a_2, \cdots, a_n) \in S^{n-1}$ der Fall.
Also ist die Zahl der Nullstellen insgesamt höchstens

$$\underbrace{d||S|^{n-1}}_{\text{Fall 1}} + \underbrace{|S|(n-1)d|S|^{n-2}}_{\text{Fall 2}}$$
$$= nd|S|^{n-1} \qquad \square$$

Dieser Algorithmus hat viele Anwendungen, z.B. Ein **Matching** in Graph $G = (V, E)$ ist $M \subseteq E$ mit $e_1, e_2 \in M \rightarrow e_1 \cap e_2 = \varnothing$
Ein **perfektes Matching** in G mit $|V| = n$ gerade ist ein Matching M mit $|M| = n/2$

Problem **PM**
geg.: Graph $G = (V, E)$
Frage: enthält G ein perfektes Matching?
(Bekannt ist $PM \in P$ (Edmonds 1965))

Der folgende Satz liefert einen einfachen Algorithmus.
Satz von Tutte
Für $G = (V, E)$ mit $V = \{1, \cdots, n\}$ definiere $n \times n$ Matrix mit $A = (a_{ij})$, wobei

$$a_{ij} = \begin{cases} x_{ij} & \text{falls } \{i,j\} \in E \text{ und } i < j \\ -x_{ij} & \text{falls } \{i,j\} \in E \text{ und } i > j \\ 0 & sonst \end{cases}$$

Dann ist $detA \not\equiv 0$ gdw G ein perfektes Matching enthält.
Beweis
$detA = \Sigma_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$
(Abkürzung $t_\sigma = \prod_{i=1}^{n} a_{i,\sigma(i)}$)
Jede Permutation zerfällt in Zyklen. Terme t_σ verschwinden, wenn für ein i $\{i, \sigma(i)\} \notin E$ ist. Übrig bleiben nur Terme t_σ, wo Zyklen von σ auch Kreise in G sind. Von solchen Termen gibt es zwei Fälle zu unterscheiden.
1. Fall σ enthält ungerade Zyklen $(i_1 i_2 \cdots i_{2k+1})$
Für solch ein σ betrachte σ', das definiert

$$\sigma'(i) = \begin{cases} \sigma(i) & \text{falls } i \notin \{i_1 \cdots i_{2k+1}\} \\ \sigma^{-1}(i) & \text{sonst} \end{cases}$$

Dann ist $t_\sigma = -t_{\sigma'}$ und $sign(\sigma) = sign(\sigma')$ Also kürzen sich alle Terme t_σ für solche Permutationen σ weg.

2. Fall σ enthält gerade Zyklen (sign(σ)=+1)

2.a $\sigma = \sigma^{-1}$, also nur 2-er Zyklen.

Jede solche σ entspricht perfektes Matching.

(perfektes Matching M: Sei $t_M := \prod_{\{i,j\} \in M} x_{ij}$

also im obigen Fall $sign(\sigma) t_\sigma = t_M^2$)

2.b $\sigma \neq \sigma^{-1}$, also σ enthält Zyklen, die länger als 2 sind.

t_σ und $t_{\sigma^{-1}}$ sind Produkt von $t_M t_{M'}$ für zwei Matchings M, M'. Dann ist

$sign(\sigma) t_\sigma + sign(\sigma^{-1}) t_{\sigma^{-1}} = 2 t_M t_{M'}$

Also insgesamt

$$detA = \Big(\sum_{M \ perfektes \ Matching \ in \ G} t_M \Big)^2$$

Also $detA \equiv 0$ gdw wenn G kein perfektes Matching enthält.

□

Teil II: Probabilistische Algorithmen

Probabilistishe Algorithmen

Definition eines deterministischen Algorithmus (nach Knuth)
1. Eine Berechnungsmethode ist ein Quadrapel (Q, I, O, f) mit
$I \subseteq Q,\ O \subseteq Q,\ f : Q \to Q$ und $f(p) = p',\ \forall p \in I$
Q: der Zustand der Berechnung
I: Eingabe
O: Ausgabe
f: Berechungsregeln.
Jedes $x \in I$ ergibt eine Folge $x_o, x_1, \cdots, x_n$, die durch
$x_{k+1} = f(x_k),\ k \geq 0,\ x_o = x$ definiert ist.
Die Folge terminiert nach k Schritten.

2. Ein Algorithmus ist eine Berechnungsmethode, die in endlich vielen Schritten für alle $x \in I$ terminiert.

3. Ein deterministischer Algorithmus ist eine formale Beschreibung für eine endliche, definite Prozedur, deren Ausgaben zu beliebigen Eingaben eindeuteig sind.

Die Hauptmotivation zur Einführung von probabilistischen Algorithmen stammt aus der Komplexitätsanalyse. Man unterscheidet zwischen dem Verhalten im schlechtesten Fall und dem Verhalten im mittleren Fall eines Algorithmus. Diese Fälle sind festgelesgt sobald das Problem und die Daten bestimmt sind.

Einige Algorithmen
Macao Algorithmus
- Mindestens bei einem Schritt der Prozedur werden einige Zahlen zufällig ausgewählt (nicht definit).
- sonst deterministisch

- Diese Algorithmen liefern immer eine korrekte Antwort. Benutzt werden sie, wenn irgenein bekannter Algorithmus zur Lösung eines bestimmten Problems im mittleren Fall viel schneller als im schlechten Fall läuft.

Beispiel für Macao Algorithmus: Nächstes-Paar

Problem: $x_1, \cdots, x_n$ seien n Punkte im k-dim. Raum R^k

gesucht: das nächste Paar (oder eins davon) x_i, x_j, so dass gilt, $d(x_i, x_j) = min\ d(x_p, x_q),\ 1 \leq p < q \leq n$, wobei d die gewöhnliche Abstandfunktion aus R^k bezeichnet.

- Man wertet alle $\frac{n(n-1)}{2}$ relevanten, gegenseitigen Abstände aus und ermittelt den minimalen Abstand.

-Dieser Algorithmus ist daher der Ordnung $O(n^2)$

- Deterministischen Algorithmen (Yuval) $\Rightarrow O(nlogn)$

Idee: Man berücksichtigt die Hüllen der Punkte $S = x_1, \cdots, x_n$ und sucht das nächste Paar innerhalb dieser Hüllen.

Schlüsselidee von Rabin

-Man wählt eine Teilmenge von Punkten zufällig aus.

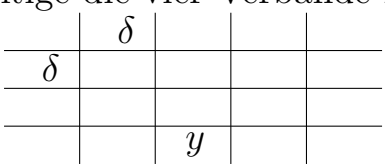
-Wähle zufällig $S_1 = \{x_{i_1}, \cdots, x_{i_m}\}$ und $m = n^{\frac{2}{3}}$, wobei n die Kardinalität von S_1 ist (= Anzahl von Elementen in S_1).

-Berechne $\delta(S_1) = min(x_p, x_q)$ für $x_p, x_q \in S_1 \rightarrow O(n)$

-Wir iterieren einmal den gleichen Algorithmus für (S_1), indem wir $S_2 \subset S_1$ mit $c(s_2) = m^{\frac{2}{3}} = n^{\frac{4}{9}}$ zufällig auswählen.

-Konstruiere einen quadratischen Verband Γ mit Netzgröße $\delta = \delta(S_1)$

-berücksichtige die vier Verbände $\Gamma_1, \cdots, \Gamma_4$.



Lemma
Gilt $\delta(s) \leq \delta(\delta$ Netzgröße von Γ ist), so existiert ein Verbandpunkt y aus Γ, so dass das nächste Paar im Quadrupel von Quadraten aus Γ direkt über und rechts von y liegt.

$\Rightarrow$ Es garantiert, dass das nächste Paar x_i, x_j aus S innerhalb eines gleichen Quadrats aus Γ_i liegt. $(1 \leq i \leq 4)$.

- Finde für jedes Γ_i die Dekomposition
$S = S_1^{(i)} \cup \cdots \cup S_k^{(i)}, \quad (1 \leq i \leq 4)$
-$\forall x_p,\ x_{q\text{-}}j^{(i)}$ berechne $d(x_p, x_q)$.
$\Rightarrow$ Das nächste Paar ist unter diesen Paaren zu finden, so dass gilt $\delta(S) = min\ d(x_p, x_q)$

Monte Carlo Algorithmus
- Gleich wie Macao Algorithmus (nicht definit)
-Zusätzlich: Ausgabe ist korrekt mit einer Wahrscheinlichkeit 1-ϵ, wobei ϵ sehr klein ist (nicht endlich)
- Diese Algorithmen liefern immer eine Antwort, wobei die Antwort nicht unbedingt richtig ist.
$\epsilon \to 0$ falls $t \to \infty$
- Das Problem bei solchen Algorithmen liegt darin, zu entscheiden, ob die Antwort korrekt ist.

Beispiel für Monte Carlo Algorithmus
(**Miller-Rabin Primzhaltest**)
Primzahltest und **Faktorisierung**
Algorithmus stellt fest, ob eine Zahl n prim ist(pseudo-prim). In diesem Algorithmus werden m Zahlen $1 \leq b_1, \cdots, b_m < n$ zufällig ausgewählt. Falls für eine gegebene Zahl n und irgendein $\epsilon > 0$, $log_2(\frac{1}{\epsilon} \leq m)$ gilt, dann wird der Algorithmus die korrekte Antwort mit einer Wahrscheinlichkeit größer als (1- ϵ) liefern.

Grundidee

Sei $W_n(b)$ die folgende Bedingung für eine ganze Zahl b:

(i) $1 \leq b < n$

(ii) (a) $b^{n-1} \not\equiv 1(mod\text{n})$, oder

(b) $\exists\ i$, so dass $2|(n-1)$ und

$1 < ggT(b^{\frac{(n-1)}{2^i}} - 1, n) < n$ gilt.

Wir bezeichnen m wie folgt: $m = \frac{(n-1)}{2^i}$

Eine ganze Zahl, die diese Bedingung erfüllt, wird Zeuge für die Teilbarkeit von n genannt.

Wenn (ii)-(a) gilt, dann ist der Fermatsche Satz verletzt.

(ii)-(b) bedeutet, dass n einen echten Teiler hat.

$\Rightarrow$ Wenn $W_n(b)$ gilt, ist n teilbar, d.h. n ist keine Primzahl.

Es stellt sich heraus, dass es aufgrund des folgenden Theorems viele Zeugen gibt, wenn n teilbar ist.

Theorem

Wenn $n > 4$ teilbar ist, dann gilt:

$\frac{3(n-1)}{4} \leq c(\{b|1 \leq b < n, W_n(b)gilt\})$,

wobei $c(S)$ die Anzahl der Elemente der Menge S ist.

Aus $b < n$ folgt,

nicht mehr als $\frac{1}{4}$ der Zahlen $1 \leq b < n$ sind keine Zeugen.

Rabin-Algorithmus

Eingabe: n ungerade ganze Zahl > 1

Ausgabe: $b = \pm 1$, falls entschieden ist, dass n prim ist.

$b = 0$, falls n teilbar ist.

1. Wähle zufällig a aus $\{1, \cdots, n-1\}$
2. Faktorisiere $(n-1)$ zu $2^l m$, mit m ungerade
3. (teste) $b := a^m \bmod n$, $i := 1$

while $b \neq -1$ **and** $b \neq 1$ **and** $i < e$

do $\{b := b^2 \bmod a, \quad i := i + 1\}$

4. (Entscheide) **if** $b = 1$ **or** $b = -1$ **then** p prim
else n zusammengesetzt $b = 0$

Bemerkung
Über Primzahltest: s. Teil III

Las-Vegas Algorithmen
- Gleich wie Macao Algorithmus (nicht definit)
- Eine Folge von zufälligen Wahlen kann unendlich sein (mit einer Wahrscheinlichkeit $\epsilon \to 0$)(nicht emdlich).
- Diese Algorithmen liefern nie eine unkorrekte Antwort, jeoch besteht die Möglichkeit, dass keine Antwort gefunden wird.

Beispiel für Las Vegas Algorithmus
(**irreduzibles Polynom in einem endlichen Körper** $\mathbb{F}_p$)
zu finden. Irreduzibel: es existiert kein Teiler, also
n irreduzibel $\Leftrightarrow \forall\ b \in \mathbb{F}_p \backslash \{1\}\ : b \nmid n$

Sei $Q(x)$ ein Polynom mit
$$Q(x) = \sum_{i=0}^{n} a_i x$$
$P(x)$ irreduzibel $\Leftrightarrow$ es existiert kein Polynom $q(x)$, so dass
$$q(x) \mid P(x)$$
Eingabe: Primzahl p und ganze Zahl n
Ausgabe: irreduzibles Polynom
repeat
1. Generiere ein zufälliges Polynom $g \in \mathbb{F}_p[x]$ n.ten Grades
2. Teste die Irreduzibilität

until test is **true**

Der Test auf Irreduzibilität basiert auf folgendem
Theorem
Seien $l_1, \cdots, l_k$ alle Primteiler von n. Bezeichne $m_i := \frac{n}{l_i}$
Ein Polynom $g(x) \in \mathbb{F}_p[x]$ vom Grad n ist irreduzibel in $\mathbb{F}_p$ gdw,

a) $g(x) \mid (x^{p^n} - x)$
b) $ggT(g(x),\ x^{p^{m_i}} - x) = 1$, für $1 \leq i \leq k$

Teil III: Primzahltests
Der Primzahltest, d.h. der Test, ob eine natrliche Zahl eine Primzahl ist, ist eines der grundlegenden Probleme der Mathematik und Informatik. Darüber hinaus gehört er zu den wichtigen algorithmischen Aufgaben mit großer praktischer Bedeutung. Das bekannteste Public-Key Kryptosystem, das RSA-System, verwendet große zufällige Primzahlen, um die Kryptanalyse zu erschweren. Hierfür generiert man eine zufällige ungerade Zahl aus einem vorgegebenen Bereich und führt den Primzahltest durch. Lange Zeit war nicht bekannt, ob es einen polynomiellen Algorithmus gibt, der den Primzahltest deterministisch entscheidet. Erst im Sommer 2002 schafften Agrawal, Kayal und Saxena den Durchbruch und konstruierten einen solchen Algorithmus. Seine Entwicklung hält man für eine der größten Errungenschaften der Algorithmik, u.a. auch wegen der Methoden, die seinem Entwurf zugrunde liegen.

Klassische Methode
Um herauszufinden, ob eine Zahl n eine Primzahl ist, liegt zunächst die klassische Methode nahe. Die Zahl n wird als zusammengesetzt indentifiziert, wenn ein Primfaktor gefunden wird. Wenn n überhaupt Primfaktoren hat, muss mindestens einer kleiner oder gleich $\sqrt{n}$ sein. Wird kein Primfaktor gefunden, ist n eine Primzahl.

Funktion classic
Eingabe: Zahl n
Ausgabe: *true* falls n zusammengesetzt ist,
false sonst.
Methode: für alle Primzahlen p mit $p \leq \sqrt{n}$

wenn $n \equiv 0 \bmod p$, dann gib *true* zurück;
gib *false* zurück.

Wie wir gesehen haben, gibt es allerdings ziemlich viele Primzahlen, die kleiner als $\sqrt{n}$ sind, nämlich $\sqrt{n}/\ln(\sqrt{n})$. Dies sind exponentiell viele im Verhältnis zur Länge von n in Bits. Alle müssen als mögliche Teiler ausgeschlossen werden. Für große Zahlen mit mehr als 100 Bit, scheidet dieses Verfahren also aus.

Fermat-Test
Überraschenderweise ist es jedoch gar nicht notwendig, einen Primfaktor der Zahl n zu kennen, um sie als zusammengesetzt identifizieren zu können. Den Ansatz dazu bietet der

Satz von Fermat
Wenn n eine Primzahl ist, dann gilt für alle $a \in \mathbb{N}$, die nicht durch n teilbar sind:

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

Man nehme also irgendeine Zahl, die nicht durch n teilbar ist, z.B. 2, und bilde

$$2^{n-1} \bmod n$$

Kommt etwas anderes als 1 heraus, so kann n *keine* Primzahl sein. Kommt 1 heraus, so ist n *ziemlich wahrscheinlich* eine Primzahl.
Im ersten Fall spielt die Zahl 2 die Rolle eines **Zeugen** dafür, dass n zusammengesetzt ist. Im zweiten Fall kann der Zeuge 2 die Zahl n nicht belasten. Somit muss n mangels Beweisen 'freigesprochen' werden. Ob n wirklich prim ist, bleibt jedoch im Prinzip zweifelhaft.

Funktion Fermat
Eingabe: Zahl n

Ausgabe: *true* falls 2 bezeugen kann, dass n zusammengesetzt ist, *false* sonst.

In Java ergibt dies folgende Implementation unter Benutzung der Funktion *modexp*:

```
boolean fermat (int n)
{
    return modexp(2, n − 1, n)! = 1;
}
```

Ergibt Fermat-Test *true*, so ist die Zahl n mit Sicherheit zusammengesetzt. Ergibt er *false*, so ist nicht ganz sicher, ob n wirklich eine Primzahl ist. Sie könnte eine *Basis-2-Pseudoprimzahl* sein. Zum Beispiel ist 561 eine solche, denn es gilt

$$2^{560} \equiv 1 \bmod 561, \quad \text{aber } 561 = 3 \cdot 11 \cdot 17$$

Derartige Zahlen sind allerdings selten; im Bereich der 100-Bit-Zahlen etwa beträgt das Verhältnis zwischen Basis-2-Pseudoprimzahlen und echten Primzahlen $1 : 10^{13}$.

Die naheliegende Idee, noch einen anderen Zeugen als nur 2 zu befragen, z. B. 3, hat in diesem Beispiel Erfolg:

$$3^{560} \not\equiv 1 \bmod 561$$

Damit ist 561 überführt, zusammengesetzt zu sein. Dies war allerdings Glück, denn es liegt in diesem Fall daran,
dass ggT(3,561)$\neq$ 1 ist. Für alle a, die teilerfremd zu 561 sind, ist 561 eine Basis-a-Pseudoprimzahl. Derartige Zahlen heißen **Carmichael-Zahlen**

Um eine Carmichael-Zahl n als zusammengesetzt zu identifizieren, müssen wir einen Zeugen a finden, der nicht teilerfremed zu n ist, also im Prinzip einen Faktor von n. Dies ist genauso schwer wie der Primzahltest von n nach der klassischen Methode.

Der Fermat-Test liefert also keine hundertprozentig sichere Aussage darüber, ob n eine Primzahl ist, außer wenn ein Aufwand in Kauf genommen wird, der genauso groß wie bei der klassischen Methode ist.

Solovay-Strassen-Test
Der Solovay-Strassen-Test (nach Robert Solovay und Volker Strassen genannt) ist ein probabilistischer Primzahltest. Er fällt in die Klasse der Monte-Carlo-Algorithmen.
Er basiert ähnlich wie der Miller-Rabin-Test auf einem Satz, mit dem festgestellt werden kann, ob eine Zahl prim ist, oder nicht.

Vorgehensweise
Der Test hat vier mögliche Ausgänge:

1. Zu einer prüfenden Zahl n, von der wir feststellen wollen ob sie eine Primzahl ist, wählen wir zufällig eine natürliche Zahl a mit $1 < a < n$.
2. Wenn nun der $\mathrm{ggT}(a, n)$ größer als 1 ist, so müssen a und n gemeinsame teiler besitzen, und n kann keine Primzahl sein.
3. Wenn diese Hürde überwunden ist, wird nach Euler
   $$j = a^{\frac{n-1}{2}} \mod n$$
   berechnet. Das ist eine etwas strengere Variante des kleinen Fermatschen Satzes.
   Wenn $j = 1$ oder $j = (n-1)$ ist, handelt sich mit einer gewissen Wahrscheinlichkeit um eine Primzahl.
   Jedoch kann es sich auch um eine Eulersche Pseudoprimzahl handeln. Im Fall, dass j weder eine 1 noch $(n-1)$ ist, handelt es sich um eine zusammengesetzte Zahl.
4. Ist auch diese Hürde genommen, dann wird das Jacobi-Symbol $J(a, n)$ berechnet.

Wenn nun $j = J(a, n)$ ist, dann ist die Wahrscheinlichkeit sehr groß, dass n eine Primzahl ist. Die Wahrscheinlichkeit für einen Irrtum beträgt maximal 50%.

Falsche Zeugen
Sei $N > 2$ eine ungerade, Nicht-Primzahl. Eine Zahl a mit ggT(a, N)=1 heißt **falscher Zeuge** für die Primalität von N bezüglich des Solovay- Strassen-Tests, falls

$$a^{\frac{N-1}{2}} \equiv (\frac{a}{N}) \pmod{N}$$

Die Menge der falschen Zeugen bildet eine Untergruppe zur multiplikativen Gruppe $(\mathbb{Z}/N)^*$ mit Ordnung $\leq \frac{1}{2}\phi(N)$
(ϕ bezeichnet die Eulersche ϕ-Funktion)
Das heißt, höchstens die Hälfte aller zur Auswahl stehender Zahlen sind falsche Zeugen.

1. Durch Zufall wurde ein Faktor gefunden und die Zahl damit als nicht prim identifiziert.
2. Das Testkriterium wird nicht erfüllt und mittels des Satzes folgt, dass es sich bei der Zahl um keine Primzahl handeln kann.
3. Das Testkriterium wird erfüllt. Bei der Zahl handelt es sich nach n erfolgreichen Durchgängen höchstens mit einer Wahrscheinlichkeit von $1/2^n$ um eine Nicht-Primzahl.