Universität Leipzig
Fakultät für Mathematik und Informatik

Deterministischer Primzahltest mit polynomieller Laufzeit: Der AKS-Primzahltest

Bachelorarbeit

vorgelegt von: Salman Salman

Betreuer: Prof. Dr. Andreas Maletti

eingereicht:

Inhaltsverzeichnis

Abstract

Primzahlen haben in der Informatik, speziell im Anwendungsgebiet der Kryptographie, eine sehr hohe Relevanz. Für moderne kryptographische Systeme ist es von Wichtigkeit selbige schnell bestimmen zu können. Hierzu werden effiziente Algorithmen zur Lösung des Primalitätsproblems(PRIMES) benötigt. Das Primalitätsproblem umfasst die Frage um die Entscheidung, ob eine gegebene Zahl eine Primzahl ist oder nicht. Der erste deterministische Primzahltest mit polynomieller Laufzeit wurde von den indischen Informatikern Agrawal, Kayal, und Saxena vorgestellt. Der nach ihnen benannte AKS-Primzahltest wird im Rahmen dieser Arbeit repräsentiert, implementiert und evaluiert.

1 Einleitung

1.1 Gliederung der Arbeit

Die Arbeit gliedert sich in sechs Bestandteile, wobei der erste Teil die Einleitung darstellt. Dort wird ein Überblick in das Thema gegeben und die Zielsetzung der Arbeit definiert.

Das zweite Kapitel enthält die Grundlagen. Hier werden Begriffe und Konzepte erläutert, die für das weitere Verständnis der Arbeit eine Rolle spielen.

Der Algorithmus und seine Korrektheit sind die Bestandteile des dritten Kapitels. Dort wird zunächst die Grundidee des AKS-Algorithmus erläutert und die nötigen Voraussetzungen des Algorithmus definiert. Des Weiteren wird der Korrektheitsbeweis des AKS-Algorithmus erklärt und durchgeführt.

Im vierten Kapitel wird die mathematische[1] Laufzeitanalyse des AKS-Algorithmus vorgestellt, dabei werden die Laufzeiten der einzelnen Schritte (STEPS 1-6) des AKS-Algorithmus analysiert. Anschließend werden mögliche Verbesserungen der Laufzeit vorgestellt.

Den Abschluss bildet die experimentelle Auswertung(Kapitel 5) der Laufzeit sowie die Zusammenfassung(Kapitel 6) dieser Arbeit.

1.2 Überblick

Eine Primzahl ist eine natürliche Zahl, die größer als 1 und ausschließlich durch sich selbst und durch 1 teilbar ist, die Menge aller Primzahlen $PRIMES = \{n \in \mathbb{N} \mid n \in \mathbb{P}\}$ heißt das Primalitätsproblem. Ein einfaches Verfahren(Algorithmus) zur Lösung des Primalitätsproblem kann aus der Definition von Primzahlen hergeleitet werden, man kann einfach testen, ob eine Zahl $m \leq \sqrt{n}$, die gegebene Zahl n teilt. Hier unterscheidet man zwischen zwei Fällen, wenn n durch m teilbar, dann ist n zusammengesetzt und sonst prim. Dieser Algorithmus heißt der Probedivision Algorithmus(engl. Trial Division Algorithm).

Soweit es bekannt ist, beschäftigten sich die Mathematiker im antiken Griechenland als erste mit den Primzahlen. Sie haben zahlreiche bedeutende Entdeckungen gemacht, beispielsweise Euklid hat bewiesen, dass es unendlich viele Primzahlen gibt.[Wikh]. Das obige Verfahren zur Lösung des Primalitätsproblems(Probedivision Algorithmus) ist auch seit der Zeit des antiken Griechenland bekannt und ist eine Verallgemeinerung von dem Sieb des Eratosthenes(engl. Sieve of Eratosthenes), mit dessen Hilfe eine Liste von Primzahlen bis zu einer vorgegeben Zahl n generiert wird.

Allerdings ist der Probedivision Algorithmus ineffizient, er braucht $\Omega(\sqrt{n})$ Schritte, um das Primalitätsproblem zu lösen. Ein effizienter Test(ein Verfahren zur Lösung des Pri-

1 Es wird in dieser Arbeit zwischen der mathematischen Laufzeitanalyse(die übliche Laufzeitanalyse, die Mithilfe der groß-O Notation beschrieben wird) und der experimentellen/empirischen Laufzeitanalyse(Ein empirischer Beweis für die Laufzeit durch Experimente).

malitätsproblem heißt ein Primzahltest oder einfach Test) soll nur polynomiell(abhängig von der Eingabegröße $= \lceil log\, n \rceil$) viele Schritte brauchen, um das Primalitätsproblem zu lösen.

Im Laufe der Jahre wurde nach einem Verfahren zum Finden von Primzahlen gesucht, im 17. und 18. Jahrhundert wurden mehrere Algorithmen zur Lösung des Primalitätsproblem vorgeschlagen. In 1770 hat der britische Mathematiker John Wilson(1741-1793) einen Satz formuliert, um Primzahlen von zusammengesetzten Zahlen zu unterscheiden. Der Satz lautet: $p \geq 2$ ist genau dann eine Primzahl, wenn $(p-1)!+1$ durch p teilbar ist.

Der französische Wissenschaftler Marin Mersenne(1588 - 1648) hat sich auch mit Primzahlen beschäftigt, er behauptete, dass für eine nach ihm benannte mersennsche Zahl(eine Zahl der Form $2^p - 1$) genau dann eine Primzahl, wenn p eine Primzahl ist. Seine Behauptung war jedoch nicht vollständig[2], beispielsweise $p = 67 \in \mathbb{P}$, aber $2^{67} - 1 \notin \mathbb{P}$. Obwohl diese Behauptung falsch war, hat sie sehr viel zur mathematischen Theoriebildung beigetragen. Zahlen für die $2^p - 1 \in \mathbb{P}$ gilt, heißen mersennsche Primzahlen, die größte Primzahl heute(Jahr: 2020) $2^{82,589,933} - 1$ ist eine mersennsche Primzahl[Wikd].

Ein anderer französischer Mathematiker Pierre de Fermat(1607-1665) hat einen Satz zur Beschreibung der Eigenschaften von Primzahlen aufgestellt. Sein Satz ist später die Grundlage für Primzahltests geworden.

Die Sätze(Algorithmen) von Wilson, Mersenne, und Fermat sind aber zum Finden von großen Primzahlen nicht geeignet und ineffizient. Natürlich hat damals sowieso niemand versucht, große Primzahlen mittels dieser Sätze zu finden, da es einfach keinen Grund dafür gab. Der frühste Einsatz von großen Primzahl findet sich im 20. Jahrhundert bei modernen kryptographische Systemen[Wikc].

Im Jahr 1997 wurde mit dem RSA-Verfahren eine Methode entdeckt, mit der man Daten sehr sicher verschlüsseln kann und die auch für den allgemeinen Gebrauch geeignet ist. Bei diesem verfahren muss aber ein Produkt aus Primzahlen berechnet werden[Wiki]. Man beachte dabei, dass die in der Praxis eingesetzte Primzahlen sehr groß sind und aus mehreren Hundert Stellen bestehen können[3]. Aus diesem Grund ist es sinnvoll einen effizienten Algorithmus(oder mehrere Algorithmen) zu haben, mit dem man solche Primzahlen "schnell" bestimmen kann[4].

Seit dem Beginn der Komplexitätstheorie in den 1960er Jahren, wurden Begriffe formalisiert und verschiedene Komplexitätsklassen definiert[AB]. Seitdem versuchten Informatiker und Mathematiker, das Primalitätsproblem in einer Komplexitätsklasse einzuordnen. Daher wurde das Primalitätsproblem auch im Laufe der Jahre intensiv untersucht. Es war trivial zu sehen, dass dieses Problem in der Komplexitätsklasse co-NP liegt[5].

2 $p = 67$ ist prim aber $2^{67} - 1 = 0\, mod\ 193707721$

3 z.B. die größte bekannte Primzahl $2^{82589933} - 1$ hat 24862048 Stellen[Wikd].

4 Schnell hier heißt nicht, dass der Algorithmus in der Praxis einsetzbar ist. Es gibt viele Algorithmen, die eine polynomielle Laufzeit haben, aber trotzdem relativ lange brauchen, um Probleme zu lösen(der AKS-Algorithmus ist ein Beispiel dafür).

5 Dies entspricht der Aussage, dass geprüft wird, ob eine Zahl n zusammengesetzt ist in NP. Man soll

In 1974 hat der australische Informatiker Vaughan Pratt eine wichtige Beobachtung gemacht, nämlich, dass das Primalitätsproblem auch in der Komplexitätsklasse NP liegt[Kur].

In 1975 hat Miller einen bedingten deterministischen Primzahltest veröffentlicht[Mil75], sein Primzahltest beruht auf dem kleinen Satz von Fermat und hat eine Laufzeit von $O(log^4 n)$. Aber er funktioniert nur unter der Annahme, dass die erweiterte Riemannsche Hypothese(ERH)wahr ist(bedingter Primzahltest[6]).

Später hat Rabin eine modifizierte Version von Millers Primzahltest veröffentlicht, die ohne ERH richtig ist, aber dadurch hat der Primzahltest seien Determinismus verloren[Wike]. Solovay und Strassen haben auch in 1977 [SS77] einen randomisierten Primzahltest mit polynomieller Laufzeit entwickelt[7]. Der Solovay-Strassen Primzahltest kann zu einem deterministischen umgewandelt werden, wenn die ERH richtig wäre.

In 1983 erzielten Adleman Pomerance, und Rumely einen großen Durchbruch, indem sie einen deterministischen Primzahltest entwickelt haben, der die Laufzeit $(log\, n)^{O(\,log\, log\, log\, n)}$ hat(die beste Laufzeit für einen deterministischen Algorithmus vor 2002).

In 1986 haben Goldwasser und Kilian einen randomisierten Primzahltest(basiert auf elliptischen Kurven) mit einer fast polynomiellen Laufzeit veröffentlicht(die Laufzeit ist nicht polynomiell für alle Inputs). Dieser Primzahltest wurde von Adleman und Huang modifiziert, um einen randomisierten Primzahltest mit polynomieller Laufzeit zu erhalten.

Das Ziel dieser Forschung war natürlich einen bedingungslosen deterministischen Primzahltest mit polynomieller Laufzeit zu entwickeln und dadurch zu zeigen, dass das Primalitätsproblem in P liegt. Leider wie bereits erwähnt, war keiner der Versuche erfolgreich, da alle entwickelten Primzahltests entweder von unbewiesenen Hypothesen, wie zum Beispiel die erweiterte Riemannsche Hypothese, abhängig waren oder probabilistisch waren[8].

In 2002 haben drei Informatiker Agrawal, Kayal, und Saxena am Indian Institute of Technology Kanpur den ersten deterministischen unbedingten Algorithmus vorgestellt, der das Primalitätsproblem in Polynomialzeit lösen kann. Das heißt sie haben gezeigt, dass das Primalitätsproblem(PRIMES) zur Komplexitätsklasse P gehört.

hier nun ein Zertifikat (Faktor) d finden und testen ob $d \mid n$ gilt. Dies ist offensichtlich in Polynomialzeit realisierbar.

6 Das Primalitätsproblem wäre in P, wenn die ERH richtig wäre, aber es gibt noch keinen Bewies für die ERH.

7 Der Algorithmus von Solovay-Strassen kann mit einer geringeren Sicherheit als der Miller-Rabin sagen, ob eine gegebene Zahl prim ist oder nicht(Fehler ist 1/4 bei Miller-Rabin, aber bei Solovay-Strassen ist der Fehler 1/2).

8 Probabilistische Algorithmen sind randomisierte Algorithmen, die auch ein falsches Ergebnis liefern können(nicht deterministisch)

1.3 Zielsetzung

In dieser Arbeit wird der AKS-Primzahltest aus dem Originalartikel([AKS02]) behandelt. Ziel dieser Arbeit ist die Korrektheit des AKS-Algorithmus zu demonstrieren und seine Laufzeit zu evaluieren. Um die Korrektheit zu zeigen, soll Zunächst der mathematische Korrektheitsbeweis durchgeführt. Des Weiteren wird die Laufzeitanalyse des AKS-Algorithmus durchgeführt, dabei soll herausgestellt werden, dass der AKS-Algorithmus eine polynomielle Laufzeit hat. Zusätzlich soll die polynomielle Laufzeit des AKS-Algorithmus durch ein Experiment illustriert werden. Schließlich wird der AKS-Algorithmus mit dem Probedivision Primzahltest verglichen[9].

9 Der Probedivision Primzahltest heißt auch der naive Primzahltest.

2 Grundlagen

In diesem Kapitel handelt es sich darum, die essenziellen Begriffe und Theoreme aus der Zahlentheorie, der Algebra und der Komplexitätstheorie, die für den AKS-Algorithmus relevant sind, zu definieren.

Bemerkung: In diesem Kapitel habe ich nur die Sätze und Theoreme bewiesen, für die (meiner Meinung nach) kein verständlicher Beweis in den Büchern vorliegt.

2.1 Zahlentheorie

In diesem Abschnitt werden grundlegende Begriffe und Theoreme aus der Zahlentheorie definiert(und ggf. bewiesen). Hier wird angenommen, das der Leser oder die Leserin eine rudimentäre Kenntnisse über die Zahlentheorie hat. Für eine ausführliche Einführung in die Zahlentheorie und Primzahlen siehe z.B. [CP05].

Definition 2.1. Seien $a, b \in \mathbb{N}$. Der **größte gemeinsame Teiler** von a und b wird mit (a, b) bezeichnet, ist die größte positive Zahl n, sodass $n \mid a$ und $n \mid b$

Definition 2.2. Zwei Zahlen $a, b \in \mathbb{N}$ heißen genau dann **Teilerfremd**, wenn $(a, b) = 1$.

Beispiel 2.1. *Der größte gemeinsame Teiler von* $24, 16$ *ist* $(24, 18) = 6$*, für zwei teilerfremde Zahlen wie* $5, 9$ *gilt* $(5, 9) = 1$*.*

Definition 2.3. Seien n, m zwei natürliche Zahlen, dann heißt die kleinste positive natürliche Zahl, die sowohl ein Vielfaches von n, als auch von m **kleinstes gemeinsames Vielfaches** beider zahlen. Hier wird das kleinste gemeinsame Vielfaches mit **kgV** bezeichnet.

Definition 2.4. Sei $n \in \mathbb{N}$, die **Primzahlzerlegung** von n ist die Darstellung der Zahl als Produkt ihrer Primfaktoren

$$n = p_1^{e_1} p_2^{e_2} ... p_M^{e_M} \tag{1}$$

Wobei e_k die Vielfachheit der Primzahl p_k ist.[10]

Definition 2.5. Seien $a, b, n \in \mathbb{N}$. a ist genau dann **kongruent** zu b modulo n, wenn $a - b \mid n$ gilt, dies wird mit $a = b \pmod{n}$ bezeichnet.

Für Primzahlen gilt die folgende Kongruenz.

10 Die Primzahlzerlegung ist eindeutig(bis auf die Reihenfolge der Primfaktoren) und existiert für jede natürliche Zahl. Diese Aussage hat Gauß in 1798 in seinem Buch Disquisitiones Arithmeticae bewiesen.[Wikb]

Theorem 2.2. *Sei n eine Primzahl, dann gilt $\binom{n}{i} = 0 \,(mod\, n)$, $\forall i \in \mathbb{N}$.*

Beweis.

$$\binom{n}{i} = \frac{(n-i-1)\cdots(n-1)\cdot n}{i!}$$

Da n im Zähler steht und n eine Primzahl ist (nicht durch Zahlen im Nenner teilbar), muss die obere Gleichung durch n teilbar sein. □

Definition 2.6. Seien $r, n \in \mathbb{N}$ mit $(n, r) = 1$, dann ist die **Ordnung von n modulo** r das kleinste k, sodass $n^k = 1(mod\ \mathrm{r})$. Die Ordnung von n modulo r wird mit $o_r(n)$ bezeichnet.

Definition 2.7. Sei $n \in \mathbb{N}$ mit $n > 1$. Die **eulersche Phi-Funktion** (mit $\phi(n)$ bezeichnet) ist die Anzahl der Zahlen k, $1 \leq k \leq n$, sodass $(k, n) = 1$. Das heißt $\phi(n) = |\{k \in \mathbb{N} \mid (k, n) = 1, 1 \leq k \leq n\}|$.[11]

Die Menge $\mathbb{Z}_n^*$ ist die Menge der teilerfremden Restklassen bezüglich eines Moduls n. Für diese Menge gilt, dass $|\mathbb{Z}_n^*| = \phi(n)$.

Theorem 2.3 (**Satz von Euler**). *Seien $a, n \in \mathbb{N}$ teilerfremd,*
dann gilt $a^{\phi(n)} = 1 \,(mod\, n)$.

Beweis. Seien $R = \{r \in \mathbb{Z}_n^* \mid (r, n) = 1\} = \{r_1, r_2, ..., r_{\phi(n)}\}$, für ein $a \in \mathbb{Z}_n^*$ gilt $\{ar_1, ar_2, ..., ar_{\phi(n)}\} = \mathbb{Z}_n^*$, dass heißt:

$$\begin{aligned} r_1 r_2 \ldots r_{\phi(n)} &= ar_1 ar_2 \ldots ar_{\phi(n)} \,(mod\, n) \\ r_1 r_2 \ldots r_{\phi(n)} &= a^{\phi(n)} r_1 r_2 \ldots r_{\phi(n)} \,(mod\, n) \\ 1 &= a^{\phi(n)} \,(mod\, n) \end{aligned}$$

Der Beweis ist auf der Argumentation der Wikipedia-Website basiert[Wika]. □

Theorem 2.4 (**Binomischer Lehrsatz**). *Sei n eine natürliche Zahl, dann gilt für $a, b \in \mathbb{Z}$:*

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}. \tag{2}$$

11 $|\cdot|$ ist die Kardinalität der Menge(Anzahl der Elemente in der Menge).

Man kann Mithilfe des binomischen Lehrsatzes folgende Gleichung herleiten.

Theorem 2.5.

$$a^n - b^n = (a-b) \cdot \sum_{k=0}^{n-1} a^k \cdot b^{n-1-k}, \ \text{für } a, b, n \in \mathbb{N}.$$

Beweis.

$$(a-b) \cdot \sum_{k=0}^{n-1} a^k \cdot b^{n-1-k} = \sum_{k=0}^{n-1} a^{k+1} \cdot b^{n-1-k} - \sum_{k=0}^{n-1} a^k \cdot b^{n-k}$$

$$= \sum_{k=0}^{n-1} (a^{k+1} \cdot b^{n-(1+k)} - a^k \cdot b^{n-k}) = a^n - b^n.$$

□

Folgende Theoreme sind für den Korrektheitsbeweis nötig:

Theorem 2.6 (**Schubfachprinzip**). *Das Schubfachprinzip(engl. pigeonhole principle) lässt sich wie folgt formulieren. Falls n Objekte auf m Mengen mit $n, m > 0$ verteilt werden und n größer als m ist, dann gibt es mindestens eine Menge, in der mehr als ein Objekt landet.*

Theorem 2.7 (**Kleiner Satz von Fermat**). *Für eine Primzahl p und ein beliebiges $a \in \mathbb{Z}$ gilt:*

$$a^p = a \, mod \, p$$

Beweis. Die Aussage kann per Induktion über $a \in \mathbb{Z}_+$ gezeigt werden.

IA: $a = 0 \Rightarrow 0^p = 0 \, mod \, p$.

IH: Nun sei die Aussage für ein beliebiges $a \in \mathbb{Z}_+$ wahr.

IS: für a + 1 gilt nach dem binomischen Lehrsatz:

$$(a+1)^p - (a+1) = \Big[a^p + \binom{p}{1} a^{p-1} + ... + \binom{p}{p-1} a + 1\Big] - (a+1).$$

Für die Koeffizienten gilt:

$$\binom{p}{k} = \frac{p \cdot (p-1) \cdots (p-k+1)}{1 \cdot 2 \cdots k}$$

$\forall k,\ 1 \leq k \leq p-1$, somit sind alle Koeffizienten(bis auf den ersten und den letzten Koeffizienten) durch p teilbar.

Also es gilt:

$$(a+1)^p - (a+1) = \left[a^p + 1\right] - (a+1) = a^p - a \, mod \, p.$$

und das ist nach der IH durch p teilbar.

□

Theorem 2.8. $\binom{2n+1}{n} > 2^{n+1}, \forall n \geq 2$

Beweis. Beweis durch vollständige Induktion:

IA: Sei $n = 2$, offensichtlich gilt $\binom{5}{2} > 2^3$.

IH: Sei nun das obere Theorem für ein beliebiges $n > 2$ wahr.

IS: Nun zeigt man, dass die Aussage auch für $n+1$ wahr ist.

$$\binom{2n+3}{n+1} = \frac{(2n+3)!}{(n+2)! \cdot (n+1)!}$$

$$= \frac{(2n+1)! \cdot (2n+2) \cdot (2n+3)}{(n+1)! \cdot (n+2) \cdot n! \cdot (n+1)}$$

$$= \frac{(2n+1)!}{(n+2)! \cdot n!} \cdot \frac{(2n+2) \cdot (2n+3)}{(n+2) \cdot (n+1)}$$

$$\underbrace{>}_{IH} 2^{n+1} \cdot \frac{(2n+2) \cdot (2n+3)}{(n+2) \cdot (n+1)} = 2^{n+1} \cdot \frac{2 \cdot (2n+3)}{(n+2)}$$

$$> 2^{n+1} \cdot \frac{2 \cdot (n+2)}{(n+2)} = 2^{n+2}$$

Die Aussage ist daher nach dem Prinzip der vollständigen Induktion wahr. □

2.2 Algebra

In diesem Abschnitt werden die für den AKS-Primzahltest nötigen Grundbegriffe aus der Algebra definiert. Dieser Abschnitt ist aber keine Einführung in die Algebra und die diskrete Mathematik. Es wird vorausgesetzt, dass der Leser oder die Leserin eine grundsätzliche Kenntnisse über die abstrakte Algebra hat.[12]

12Für eine Einführung in die Algebra und die diskrete Mathematik siehe: [Lid86] oder [Pin].

2.2.1 Algebraische Strukturen

Gruppen und Untergruppen

Eine Gruppe ist eine fundamentale algebraische Struktur, eine Gruppe besteht aus einer Menge von Elementen und einer zweistelligen Verknüpfung. Für alle Elemente einer Gruppe müssen die sogenannten Gruppenaxiome gelten.

Definition 2.8. Eine **Gruppe** ist ein Tupel $(G, \circ)$ bestehend aus einer Menge G und einer inneren zweistelligen Verknüpfung $\circ$[13], für die folgende Eigenschaften gelten:

- **Abgeschlossenheit:** Für alle Gruppenelemente $a, b \in G$ gilt $a \circ b \in G$.
- **Assoziativität:** Für alle Gruppenelemente a, b und c gilt:
  $(a \circ b) \circ c = a \circ (b \circ c)$.
- **Neutrales Element:** Es gibt ein einziges Neutrales Element $e \in G$, mit dem für alle Gruppenelemente $a \in G$ gilt:
  $a \circ e = a = e \circ a$.
- **Inverses Element:**Zu jedem Gruppenelement $a \in G$ existiert ein einziges inverses Element $a^{-1} \in G$, sodass $a^{-1} \circ a = e = a \circ a^{-1}$.

Bemerkung: Wenn nur die Assoziativität Eigenschaft gilt, und ein neutrales Element existiert, dann heißt die Struktur heißt ein Monoid[14].

Beispiel 2.9. *Die Menge der ganzen Zahlen $\mathbb{Z}$ mit der Addition $+$ bildet eine Gruppe. Für alle $a, b, c \in \mathbb{Z}$ gilt:*

1. *$a + b \in \mathbb{Z}$ (Abgeschlossenheit).*
2. *$(a + b) + c = a + (b + c)$ (Assoziativität).*
3. *$\forall a \in \mathbb{Z}$, $a + 0 = a = 0 + a$ (Neutrales Element).*
4. *$\forall a \in \mathbb{Z}$, $a + (-a) = 0 = (-a) + a$ (Inverses Element).*

Die obere Gruppe im Beispiel heißt auch eine kommutative oder abelsche Gruppe, da $a + b = b + a$, $\forall a, b \in \mathbb{Z}$ gilt. Diese Eigenschaft ist aber offensichtlich nicht für alle Gruppen erfüllt, ein Beispiel für eine nicht abelsche Gruppe ist die symmetrische Gruppe S_3[Wikj].

Definition 2.9. Eine Gruppe $(G, \circ)$ heißt **kommutativ** oder **abelsch**, wenn

$$a \circ b = b \circ a$$

$\forall a, b \in G$ gilt.

Beispiel 2.10. *Die Gruppe $(\mathbb{Z}, +)$ ist eine abelsche Gruppe, da die Operation $+$ kommutativ ist.*[15]

13. Die zweistellige Operation $\circ$ ist durch $\circ : A \times A \to A$ definiert, dabei ist A eine beliebige Menge.

14Diese Struktur wird im Abschnitt der Komplexitätstheorie verwendet

15 kommutativ bzw. abelsch heißt, dass die Argumente einer Operation vertauscht werden können, ohne dass sich das Ergebnis verändert.

Eine wichtige Gruppe ist die multiplikative Gruppe $\mathbb{K}^* = \mathbb{K} - \{0\}$[16] der Primrestklassen. Für eine Primzahl p besteht diese Gruppe aus den Elementen $\{0, 1, \ldots, p-1\}$.

Definition 2.10. Die Gruppe der **Primrestklassen** bezüglich eine Moduls einer ganzen Zahl n besteht aus den Restklassen $a + n\mathbb{Z}$, deren Elemente teilerfremd zu n sind. Mit anderen Worten ist diese Menge, die Menge aller Zahlen $a \in \mathbb{Z}$ für die $(a, n) = 1$. Sie wird mit $\mathbb{Z}_n^*$ bezeichnet, mit $\mathbb{Z}_n^* = \{a \in \mathbb{Z} \mid (a, n) = 1\}$.

Definition 2.11. Sei $(G, \circ)$ eine Gruppe. Eine Menge $H \subseteq G$ heißt eine **Untergruppe** von G, wenn H mit der Operation $\circ$ eine Gruppe bildet.

Ordnung

Ein weiteres wichtiges Konzept der Gruppentheorie ist die Ordnung. Man unterscheidet zwischen der Ordnung einer Gruppe, und der Ordnung eines Elementes von einer beliebigen Gruppe. Die Ordnung einer Gruppe G ist die Kardinalität dieser Gruppe(Anzahl der Elemente in der Gruppe). Während Die Ordnung eines beliebigen Elements $a \in G$ die kleinste Zahl n, sodass $a^n = e$, ist.[17]Wobei e das neutrale Element der Gruppe G ist.

Definition 2.12. Die **Ordnung** einer Gruppe $(G, \circ)$ ist die Kardinalität dieser Gruppe. Das heißt die Ordnung der Gruppe ist die Anzahl der Elemente in G. Die Ordnung einer Gruppe G wird mit $|G|$ bezeichnet.

Definition 2.13. Sei $(G, \circ)$ eine Gruppe, dann heißt die kleinste Zahl n, sodass $a^n = a \circ a \circ \cdots \circ a = e$ die **Ordnung** von a und wird mit $ord_G(a) = n$ bezeichnet.

Zyklische Gruppen

Eine zyklische Gruppe G ist eine Gruppe, die von einem einzigen $a \in G$ Element erzeugt wird. Zunächst betrachtet man die Potenzen der Elemente von beliebigen Gruppen. Sei $(G, \circ)$ eine Gruppe und $a \in G$ beliebig. Wir betrachten die Menge $\langle a \rangle = \{a \circ a \circ \cdots \circ a = a^i \mid i \in \mathbb{Z}\}$, diese Menge heißt die Menge der Potenzen von $a \in G$. G heißt zyklisch, wenn sie ein Element a enthält, sodass die Potenzen von a die Gruppe G erzeugen.

Definition 2.14. Eine Gruppe G[18] ist **zyklisch**, wenn sie ein Element a enthält, sodass jedes Element von G eine Potenz von a ist. Das heißt $\langle a \rangle = G$.

Ringe

Viele Strukturen verfügen nicht nur über eine Operation, sondern über zwei, die zueinander in Beziehung stehen. Das fuhrt zum Begriff des Ringes

16 Hier entspricht der Mengendifferenz-Operator, das heißt $\mathbb{K} - \{0\} = \mathbb{K} \setminus \{0\}$

17 a^n ist keine Potenzierung sondern eine wiederholte Anwendung(in diesem Fall n mal) des Gruppen Operators. z.B. für die Gruppe $(\mathbb{Z}, +)$ ist $a^n = \underbrace{a + a + \ldots a}_{n-mal}$

18 G ist eine Abkürzung für $(G, \circ)$.

Definition 2.15. Ein **Ring** $(R, +, \cdot)$ ist eine Menge R mit zweistelligen Operationen $+, \cdot$, sodass diese folgenden Gesetze gelten:

- $(R, +)$ ist eine abelsche Gruppe.
- $\cdot$ ist assoziativ, das heißt $a \cdot (b \cdot c) = (a \cdot b) \cdot c$, $\forall\, a, b, c \in R$.
- Das Distributivgesetz gilt für alle $a, b, c \in R$, das heißt: $a \cdot (b + a) = a \cdot b + a \cdot c$, $\forall a, b, c \in R$.

Definition 2.16. Ein Ring R heißt **kommutativ**, wenn die Operation $\cdot$ kommutativ ist.[19]

Ein bekannter Ring ist der Ring der ganzen Zahlen modulo n(spielt eine große Rolle für den Korrektheitsbeweis des AKS-Algorithmus).

Der Ring modulo n ist wie folgt definiert:

$$\mathbb{Z}_n = \{1, 2, \ldots, n-1\}$$

Die Addition für den Ring $\mathbb{Z}_n$ Nun kann die Addition für den Ring $\mathbb{Z}_n$ definiert:

$$a + b = a + b \, mod \, n, \; \forall a, b \in R.$$

Dies kann am besten mit einem Beispiel illustriert werden. Sei $n = 6$, dann gilt:

$$1 + 5 = 6 \; mod \, 6 = 0, \;\; 4 + 5 = 9 \; mod \, 6 = 3.$$

Das gleiche gilt auch für die Multiplikation:

$$a \cdot b = ab \; mod \, n$$

Sei n wieder gleich 6, dann gilt das folgende für die Multiplikation:

$$4 \cdot 3 = 12 \; mod \, 6 = 0$$

Bemerkung: $\mathbb{Z}_n$ ist ein kommutativer Ring.

Definition 2.17. Ein Ring heißt **nullteilerfrei**, wenn die Multiplikation von zwei Elementen $a, b \neq 0 \in R$ nicht null ist. Das heißt $a \cdot b \neq 0$, wenn $a, b \neq 0$.

Definition 2.18. Ein nullteielrfreier Ring heißt **Integritätsring**, wenn er kommutativ und von Null verschieden$(R \neq 0)$ ist.

Definition 2.19. Sei I eine Teilmenge des Rings $(R, +, \cdot)$. I heißt **Ideal**, wenn gilt:

19 z.B. wenn die Operation $\cdot$ die übliche Multiplikation ist. Dies ist nicht immer zwangsweise der Fall. Eine nicht kommutative Multiplikation wäre z.B. die Matrizenmultiplikation.

1. $e \in I$. (es existiert ein neutrales Element)
2. $\forall a, b \in I$ ist $a + b \in I$.
3. $\forall a \in I$ und $r \in R$ ist $r \cdot a \in I$.
4. $\forall a \in I$ und $r \in R$ ist $a \cdot r \in I$.

Definition 2.20. ist $(R, +, \cdot)$ ein Ring und I ein Ideal von R, dann bildet die Menge $R/I = \{a + I \mid a \in R\}$ der Äquivalenzklassen modulo I mit folgenden Verknüpfungen einen Ring:

- $(a + I) + (b + I) = (a + b) + I$.
- $(a + I) \cdot (b + I) = (a \cdot b) + I$.

Diesen Ring nennt man **Quotientenring**.

2.2.2 Polynome

Ein Polynom ist ein Ausdruck der Form $a_0 + a_1X + \cdots + a_nX^n$. Die a_i, $i \in \{1, 2, \ldots, n\}$ heißen die Koeffizienten des Polynoms. Polynome sind in der Regel in einem algebraischen Ring definiert.

Definition 2.21. Sei R ein beliebiger Ring. Ein **Polynom** $f(X)$ über dem Ring R hat folgende Form:

$$f(X) = \sum_{i=0}^{n} a_i X^i = a_0 + a_1X + \cdots + a_nX^n.$$

Die Arithmetik von Polynomen ist ähnlich zur Arithmetik von ganzen Zahlen. Die Operationen $+, \cdot$ und die Menge der Polynome mit Koeffizienten $a \in R$ bilden einen Ring, da $f(X) + g(X) = \sum_{i=0}^{n} a_i X^i + \sum_{i=0}^{n} b_i X^i = \sum_{i=0}^{n} (a_i + b_i) X^i$ und $f(X) \cdot g(X) = \sum_{i=0}^{n+m} c_i X^i$. Für eine ausführliche Beschreibung von Polynomarithmetik über einem Ring siehe bspw. [Lid86].

Definition 2.22. Seien R ein Ring und $fR[X]$. Dann heißt ein Element $a \in R$ eine **Wurzel** oder Nullstelle des Polynoms f, wenn $f(a) = 0$ gilt.[20]

Definition 2.23. Sei R ein Ring, dann ist der **Polynomring** $R[X]$ die Menge aller Polynome der Form $a_0 + a_1X + a_2X^2 + ... + a_nX^n$, wobei $a_0, a_1..., a_n \in R$.

Definition 2.24. Sei $f = (a_0, a_1, ...a_n) \in R[X]$, $f \neq 0$ ein Polynom, dann heißt

$$deg\, f = max\{i \mid a_i \neq 0\}$$

der **Grad** vom Polynom f.

[20] $f(a)$ bedeutet, dass X durch a ersetzt wird(X ist eine Variable).

Definition 2.25. Sei R ein Integritätsring. Dann heißt ein Polynom $f \in R[X]$ **irreduzibel**, wenn $f \neq 0$ nicht invertierbar(es existiert kein $f^{-1} : f \cdot f^{-1} = e$, wobei e das neutrale Element von R ist) in $R[X]$ ist und für $g, h \in R[X]$ und $f = gh$ entweder g oder h invertierbar ist.[21]

Theorem 2.11. *Sei R ein kommutativer Ring und $f \in R[X]$ ein Polynom, sodass $f \neq 0$. Ferner seien $a_1, a_2, ..., a_n \in R$ die Wurzeln von f mit $a_i - a_j \in R \setminus \{0\}$, $\forall i, j,\ 1 \leq i, j, \leq n$. Dann ist $deg\, f \geq n$.*

Beweis. Sei $deg\, f = n - 1,\ n \in \mathbb{N}$, wobei $f(X) = b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + ... + b_1X + b_0$ mit $b_{n-1} \neq 0$. Weiterhin seien $a_1, a_2, ..., a_n$ die Wurzeln des Polynoms $f(X)$, dann gilt $f(a_i) = 0$, $\forall 1 \leq i \leq n$, das heißt $b_{n-1}a_i^{n-1} + ... + b_1a_i + b_0 = 0$, $\forall 1 \leq i \leq n$.

Zudem sei $A = [a_{i,j}]_{1 \leq i,j \leq n}$ die Matrix der Wurzeln und $b = (b_1b_2...b_n) \neq 0$ der Vektor der Koeffizienten von $f(X)$. Dann gilt $b \cdot A = 0$, aber $det\, A \neq 0 \Rightarrow b = 0$ und dies ist ein Widerspruch(Da $b \neq 0$), daraus folgt es existiert $f(X) : deg\, f \geq n$ □

Polynomdivision

Im folgenden werden Polynome über (nicht zwangsweise endliche) Körper betrachtet. Ein Polynom $g \in \mathbb{K}[X]$ teilt das Polynom $f \in \mathbb{K}[X]$, wenn ein drittes Polynom $h \in \mathbb{K}[X]$ existiert, sodass $f = gh$.

Theorem 2.12. *Sei $g \neq 0$ ein Polynom im Körper $\mathbb{K}[X]$, dann existiert für jedes Polynom $f \in \mathbb{K}[X]$ Polynome $q, r \in \mathbb{K}[X]$, sodass*

$$f = gq + r, wobei\ deg\, r < deg\, g.$$

erfüllt ist.

Beispiel 2.13. *Seien $f(X) = 2X^5 + X^4 + 4X + 3 \in \mathbb{K}_5[X]$, $g(X) = 3X^2 + 1 \in \mathbb{K}_5[X]$, wobei $\mathbb{K}_5$ der Körper der Restklassen von 5 ist[22]. Gemäß Theorem 2.12 existieren $q, r \in \mathbb{K}_5[X]$, sodass $f = qg + r$, man verwendet nun die schriftliche Division Algorithmus(in $\mathbb{K}_5$)[Wikk]:*

$$\underbrace{\frac{2X^5 + X^4 + 4X + 3}{3X^2 + 1}}_{\frac{f}{g}} = \underbrace{(4X^3 + 2X^2 + 2x + 1)}_{q} + \underbrace{\frac{2X + 2}{3X^2 + 1}}_{\frac{r}{g}}$$

Folglich ist $q(X) = 4X^3 + 2X^2 + 2X + 1,\ r(X) = 2X + 2$ und $deg(r) = 1 < deg(g) = 2$.

21 Für mehrere Beispiele und Definitionen von Invertierbaren Elementen bzw. Polynomen siehe : `https://de.wikipedia.org/wiki/Inverses_Element`.

22 $\mathbb{K}_5 = \{0, 1, 2, 3, 4\}$

2.2.3 Körper

Eine zusätzliche algebraische Struktur ist der Körper. Im Korrektheitsbeweis werden (zyklotomische)Polynome über endlichen Körpern behandelt. Aus diesem Grund sind die Struktur und ihre Eigenschaften für das weitere Verständnis der Arbeit nötig, insbesondere, dass die multiplikative Gruppe von endlichen Körpern zyklisch ist.

Definition 2.26. Ein **Körper** $\mathbb{K}$ ist eine Menge mit zwei binären Operation $+, \circ$ (die Addition und Multiplikation genannt werden) für die folgende Bedingungen erfüllt sind:

1. $(\mathbb{K}, +)$ ist eine abelsche Gruppe(neutrales Element 0).
2. $(\mathbb{K} \setminus \{0\}, \circ)$ ist eine abelsche Gruppe (Identitätselement 1).
3. Distributivgesetze:
   $a \circ (b + c) = a \circ a + a \circ c$, $\forall a, b, c \in \mathbb{K}$.
   $(a + b) \circ c = a \circ c + b \circ c \forall a, b, c \in \mathbb{K}$

Eigenschaften von endlichen Körpern

Ein Körper heißt endlich, wenn die Grundmenge $\mathbb{K}$ endlich ist($|\mathbb{K}| < \infty$). Für jede Primzahl p ist der Ring $\mathbb{K}_p$ ein Körper mit p Elementen(Die Restklassen mod p), dieser Körper heißt auch Galois Körper der Ordnung p.

Theorem 2.14. *Sei p eine Primzahl, dann ist der Ring $\mathbb{Z}_p$ ein Körper.*

Eine zentrale Eigenschaft eines beliebigen endlichen Körpers $\mathbb{K}_p$ mit p Elementen ist, dass die Multiplikative Gruppe $\mathbb{K}_p^* = \mathbb{K} - \{0\}$ zyklisch ist.

Theorem 2.15. *Sei $\mathbb{K}_p$ ein Körper, dann ist $\mathbb{K}_p^*$ eine zyklische Gruppe. Das heißt es existiert ein $g \in \mathbb{K}_p^*$ mit $\mathbb{K}_p^* = \{1, g, \ldots, g^{|\mathbb{K}|-2}\}$. Für einen Beweis siehe [Die73] oder [Lid86].*

Theorem 2.16. *Sei $\mathbb{K}$ ein Körper mit m Elementen, dann gilt:*

$$a^{m-1} = 1$$

für alle $a \in \mathbb{K}^x$.[23]

Theorem 2.17. *Seien $\mathbb{K}$ ein Körper und $f \in \mathbb{K}[X]$ ein Polynom. Der Restklassenring $\mathbb{K}/f(X)$ ist genau dann ein Körper, wenn f in $\mathbb{K}$ irreduzibel ist.*

Theorem 2.18. *Sei $\mathbb{K}$ ein Körper und $a \in \mathbb{K} - \{0\}$. Weiterhin sei $n \in \mathbb{N}$, mit $n \geq 1$. Dann gilt $a^n = 1 \Leftrightarrow ord_{\mathbb{K}}(a) \mid n$.*

Für einen Beweis für Theorem 2.16,Theorem 2.17 und Theorem 2.18 siehe [Lid86].

22Für den Beweis siehe [Lid86].

23 Für einen Beweis siehe bswp. `http://www.heldermann-verlag.de/Mathe-Kabinett/Kreisteilungspolynome.pdf`.

2.2.4 Zyklotomische Polynome

Das n-te zyklotomische Polynome (auch: Kreisteilungspolynom) ist ein spezielles irreduzibles Polynom mit ganzzahligen Koeffizienten, die Teiler von $X^n - 1$ sind. Die Wurzeln des n-ten zyklotomischen Polynom sind alle n Einheitswurzeln.

Definition 2.27. Sei n eine natürliche Zahl, die n-te **Einheitswurzel** wird mit ζ bezeichnet ist eine komplexe Zahl, sodass

$$\zeta^n = 1.$$

Z.B. 1 und -1 sind die quadratischen Einheitswurzeln, und 1, -1, i, -i, sind die Einheitswurzeln für $n = 4$.

Definition 2.28. Eine n-te Einheitswurzel ζ heißt eine **primitive Einheitswurzel**, wenn alle n-ten Einheitswurzeln als Potenzen von ζ darstellbar sind.

Theorem 2.19. *Die Ordnung einer primitiven n-ten Einheitswurzel ζ ist n.*

Theorem 2.20. *Sei $n \in \mathbb{N}$ existieren genau $\phi(n)$ primitive Einheitswurzeln.*[24]

Für eine Menge von primitiven Einheitswurzeln $\zeta_1, \zeta_2, \ldots, \zeta_{\phi(n)}$ kann das n-te zyklotomische Polynom $\Phi_n(X)$ als ein Produkt der $\phi(n)$ primitive Einheitswurzeln dargestellt werden. Das heißt $\Phi_n(X) = (X - \zeta_1) \cdot (X - \zeta_2) \cdots (X - \zeta_{\phi(n)})$.

Definition 2.29. Das n-te **zyklotomische Polynom** Φ_n ist durch:

$$\Phi_n(x) = \prod_{\substack{1 \le k \le n \\ (k,n)=1}} \left(x - \zeta^k\right). \tag{3}$$

definiert.

Theorem 2.21. *Für jede natürliche Zahl n . gilt:*

$$x^n - 1 = \prod_{d|n} \Phi_d(n). \tag{4}$$

Für den Beweis, siehe [bar].

24Dies folgt der Definition der primitiven Einheitswurzeln und aus der Tatsache, dass es für eine Zahl n genau $\phi(n)$ teilerfremde Zahlen (im Intervall $[1, n]$) gibt

2.3 Komplexitätstheorie

In diesem Abschnitt handelt es sich darum, die relevanten Begriffe aus dem Gebiet der Komplexitätstheorie, deutlich zu definieren. Grundlagen der theoretischen Informatik, wie Turingmaschinen, groß O-Notation, und Entscheidungsprobleme werden in dieser Arbeit nicht definiert, jedoch sind sie eine Voraussetzung, um die hier behandelten Themen zu verstehen[25]. Anschließend werden Fakten über die Komplexität der einfachen Operationen(Addition, Subtraktion, Multiplikation,...) im Computer dargestellt.

Der AKS-Algorithmus ist hauptsächlich dafür bekannt, eine polynomielle Laufzeit zu haben. Daher ist zuerst wichtig, das Konzept der polynomiellen Laufzeit zu definieren. Ein Algorithmus mit polynomieller Laufzeit ist einer, dessen Laufzeit durch eine Polynomfunktion der Größe ihrer Eingabe begrenzt ist.

Definition 2.30. In der Komplexitätstheorie ist **P** die Komplexitätsklasse, die alle Entscheidungsprobleme enthält, die in polynomiellzeit für deterministische Turingmaschinen lösbar sind.

Neben der Komplexitätsklasse P gibt es auch weitere Klassen, wie z.B. NP und co-NP, diese Klassen werden hier aber nicht behandelt, da sie nicht direkt mit dem AKS-Algorithmus verbunden sind. Wie bereits erwähnt, ein Problem ist in P, wenn es von einem Algorithmus mit polynomieller Laufzeit deterministisch gelöst wird. Im Rahmen von Primalitätstesten bedeutet dies ein Algorithmus, der eine natürliche Zahl n als Eingabe verwendet und bestimmt, ob n eine Primzahl ist oder nicht. Das heißt, dass die Eingabegröße ungefähr $log\, n$ ist.[26] In dieser Arbeit wird die Eingabegröße[27] mit $\|n\|$ bezeichnet.

Definition 2.31. Für eine natürliche Zahl $n \geq 1$, ist $\|n\| = \lceil log(n+1) \rceil$, wobei $log\, n = log_2 n$

Ein weiteres wichtiges Mittel in der Laufzeitanalyse ist die groß O-Notation, die übliche Landau O-Notation sollte für den Leser bekannt sein. Hier wird jedoch eine neue Notation $O^{\sim}$ eingeführt.

Definition 2.32. Eine Funktion $f(n)$ ist in $O^{\sim}(t(n))$, wenn

$$f(n) = O(t(n) \cdot poly(log\, t(n)))^{28}$$

Beobachtung:

$$O^{\sim}(log^k n) = O((log^k n) \cdot poly(log\, log^k n)) = O((log^k n) \cdot poly(log(k \cdot log n))) = O(log^{k+\epsilon} n)$$

25 Für eine Einführung in die Berechenbarkeit und die Komplexitätstheorie siehe: [Pap93],[AB].

26 Da die Zahlen im Computer im binären System dargestellt werden, ist die Eingabe ungefähr $log_2(n)$(Anzahl der Bits, um n zu repräsentieren). Für weitere Information zu diesem Thema siehe: [Die73], [GG99].

27 In dieser Arbeit wird auch das Synonym Inputgröße manchmal benutzt.

Für alle $\epsilon > 0$, wenn ein Primzahltest eine Laufzeit von $O^{\sim}(log^k n)$ hat. Dann ist die Laufzeit dieses Primzahltests polynomiell von $log\, n$ abhängig. Das heißt dieser Primzahltest hat eine polynomielle Laufzeit.

Komplexität der einfachen Computeroperationen

Um die Laufzeitanalyse des AKS-Algorithmus zu verstehen, sind fundamentale Fakten über Computer-Arithmetik und die Komplexität der einfachen Operationen, wie Additionen und Multiplikationen nötig.

Fakt 2.22. *Seien $n, m \in \mathbb{N}$:*

1. *Addition und Subtraktion von n und m können in $O(\|n\| + \|m\|) = O(log\, n + log\, m)$ Bit-Operationen berechnet werden.*
2. *Multiplikation $n \cdot m$ kann in $O(\|n\| \cdot \|m\|) = O(log\, n \cdot log\, m)$ Bit-Operationen berechnet werden.*
3. *Modulo-Berechnung von n mod m kann in $O(\|n\| - \|m\| + 1)$ Bit-Operationen berechnet werden.*

Fakt 2.23. *Seien $n, m \in \mathbb{N}$, $\|n\| = \|m\| = k$*

1. *Multiplikation kann in $O(k(log\, k)(log\, log\, k)) = O^{\sim}(k)$ Bit-Operationen berechnet werden.*
2. *n mod m kann in $O(k(log\, k)(log\, log\, k)) = O^{\sim}(k)$ Bit-Operationen berechnet werden2.*

Theorem 2.24. *Sei (M,∘,1) ein Monoid[29], dann existiert ein Algorithmus für alle $a \in M$ und $n \geq 0$, sodass a^n in $O(log\, n)$ Multiplikationen in M berechnet wird.[30]*

29Ein Monoid ist eine algebraische Struktur, wo $\circ$ assoziativ ist und M ein neutrales Element enthält[Wikf].

30 Algorithmus 3 kann a^n in $2 \cdot \|n\| = O(log\, n)$ Multiplikationen in M berechnen

3 Der AKS-Primzahltest

3.1 Grundidee des Algorithmus

Die Idee des Algorithmus ist basiert auf Verallgemeinerung des kleineren fermatschen Satz (Theorem 2.7) für Polynome und beruht auf folgender Identität.

Lemma 3.1. *seien $a, n \in \mathbb{N}$ mit $a < n$ und teilerfremd(Def. 2.2), dann ist n genau dann eine Primzahl, wenn*

$$(X + a)^n = X^n + a(mod\ n).$$

erfüllt ist, dabei ist X ein Polynom über dem Ring $\mathbb{Z}_n[X]$

Beweis. Aus dem binomischen Lehrsatz(2.4) folgt, dass der Koeffizient von X^i in dem Polynom $(X + a)^n$ gerade $\binom{n}{i}a^{n-i}$ ist.[31]

" $\Rightarrow$ "

Sei n eine Primzahl, dann ist es nach Theorem 2.2 klar, dass $\forall i,\ 0 < i < n$,

$\binom{n}{i} = \frac{n!}{(n-i)!\cdot i!} = 0 \pmod n$. Das heißt alle Koeffizienten (bis auf den ersten und den letzten) sind Null.

Für $i = 0$ erhält man $\binom{n}{0}a^n X^0 = a^n$, analog für $i = n$ gilt $\binom{n}{n}a^0 X^n = X^n$. Daraus folgt:

$$(X + a)^n = a^n + 0 + 0 + ... + 0 + X^n = a^n + X^n \underbrace{=}_{2.7} X^n + a\ (\ mod\ n).$$

" $\Leftarrow$ "

Sei n nun eine zusammengesetzte Zahl(COMPOSITE). Man betrachtet einen Faktor q von n, mit der Vielfachheit k(Dabei ist zu beachten, dass Für $1 < q < n$, $q^k | n$, aber $q^{k+1} \nmid n$).

Der Koeffizient von X^q sieht wie folgt aus:

$$\binom{n}{q} \cdot a^{n-q} = \frac{n!}{(n-q)! \cdot n!} \cdot a^{n-q} = \frac{n(n-1)\cdots(n-q+1)}{q!} \cdot a^{n-q}.$$

[31] $(X + a)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} \cdot X^i$.

Im Nenner lässt sich $q!$ als $q \cdot (q-1)!$ schreiben und im Zähler lässt sich n als $q^k \cdot m$ schreiben, $m \in \mathbb{Z}_+$. Das q im Nenner hebt sich mit einem der q Werte im Zähler auf. Der resultierende Ausdruck ist daher nicht durch q^k teilbar, außerdem sind q^k und a^{n-k} teilerfremd. Daraus folgt: $(X+a)^n \neq X^n + a(\text{mod } n)$. □

Es wäre nun möglich anhand dieser Identität einen Primzahltest für eine Zahl n zu realisieren. Dies wäre jedoch sehr ineffizient, da im Polynom die Auswertung von n Koeffizienten nötig ist. Mit anderen Worten der Algorithmus braucht $\Omega(n)$ um zu entscheiden, ob die Zahl n eine Primzahl ist oder nicht, und das ist nicht in Polynomialzeit realisierbar.

Die Idee von AKS ist nicht nur modulo n, sondern auch modulo ein Polynom $(X^r - 1)$ zu nehmen. Das bedeutet, dass nicht nur alle Koeffizienten c_k durch c_k modulo n ersetzt werden, sondern auch jedes X^k durch $X^{k \, mod \, r}$. Das heißt es wird in dem Fall mit Polynomen vom Grad weniger als r gearbeitet. Die Rechenkosten bleiben im polynomiellen Bereich, solange r höchstens $O((logn)^c)$, $c > 0$ ist. Daher ist das Hauptziel jetzt ein entsprechend kleines r zu wählen und zu testen, ob die Gleichung:

$$(X+a)^n = X^n + a(\text{mod } X^r - 1, n) \tag{5}$$

für $a, r \geq 1$ erfüllt ist.

Nach Lemma 3.1 ist die Gleichung (5) für alle Primzahlen erfüllt. Aber ein Problem dabei ist, dass es auch einige zusammengesetzte Zahlen gibt, für die Gleichung (5) erfüllt ist. Die Idee vom AKS-Algorithmus ist ein "geeignetes" r zu wählen und nur für dieses r zu testen ob (5) für eine "kleine" Anzahl von a Werten erfüllt ist. Wenn n zusammengesetzt ist, dann existiert ein Paar (a, r), für es Gleichung (5) nicht erfüllt ist. Um einen deterministischen Algorithmus mit polynomieller Laufzeit zu erhalten, sollen a und r von der Eingabegröße($\sim log\, n$) polynomiell abhängig sein.

3.2 Zur Wahl von a und r

In diesem Abschnitt wird die Auswahl des geeigneten r und die Obere Schranke für die Anzahl der zu testenden a Werte untersucht. In der Einleitung des AKS-Algorithmus wurde erwähnt, dass man ein geeignetes r findet, und für das r test, ob

$$(X+a)^n = X^n + a\,(mod\, X^r - 1, n) \tag{6}$$

für mehrere Werte von a erfüllt ist.

Dies ist aber nicht immer ausreichend um die Primalität einer Zahl zu zeigen, da es auch zusammengesetzte Zahlen gibt, die Gleichung (6) erfüllen. Solche zusammengesetzte Zahlen sollen mithilfe von Bedingungen an r, a identifiziert werden können.

Wenn eine Zahl n zusammengesetzt ist, dann findet man ein Paar (a, r)[32], für es der Algorithmus COMPOSITE für dieses n liefert.

Definition 3.1 (**AKS-Zertifikat**)**.** Wenn der AKS-Algorithmus für eine Zahl n ein Paar von Zahlen (a, r), $a, r \in \mathbb{N}$ findet, sodass der Algorithmus COMPOSITE liefert. Dann heißt das Paar (a, r) ein AKS-Zertifikat(ein Zertifikat, dass n zusammengesetzt ist).

Wie schon angedeutet, es gibt zusammengesetzte Zahlen für die

$$(X + a)^n = X^n + a \,(mod\, X^r - 1, n)$$

erfüllt ist. Das heißt es gibt zusammengesetzte Zahlen für die man ohne weitere Bedingungen an r und a kein AKS-Zertifikat finden kann. Hier stellt sich die Frage, ob man Bedingungen finden kann, wenn sie für ein r(ein geeignetes r) in $O(log^k n)$, $k > 0$ und $0 < a < \ell$ erfüllt sind, dann ist n sicherlich eine Primzahl.

Es wird zunächst angenommen, dass n eine Potenz einer Primzahl p ist(p ist ein Primfaktor von n). Danach wird versucht, Bedingungen für a, r, p, n zu finden, um n zu zwingen, eine Primzahlpotenz zu sein. Wenn n unter diesen Bedingungen keine Primzahlpotenz ist, dann ist n eine Primzahl, das heißt $n = p^1$(wenn n schon eine Primzahlpotenz ist, dann ist n trivialerweise zusammengesetzt).

3.2.1 Introspektivität

Für die Bedingungen für r und a wird zuerst die Introspektivität Eigenschaft definiert.

Definition 3.2. Seien $f(X)$ ein Polynom in $\mathbb{Z}[X]$ und $m \in \mathbb{N}$, m ist **introspektiv** bezüglich $f(X)$, wenn

$$[f(X)]^m = f(X^m) \text{ (mod } X^r - 1, p)$$

gilt.

Lemma 3.2. *Seien m, m' introspektive Zahlen bezüglich eines Polynoms $f(X)$ in $\mathbb{Z}[X]$. Dann ist $m \cdot m'$ auch introspektiv bezüglich $f(X)$.*

Beweis. Da m bezüglich $f(X)$ introspektiv ist, gilt:

$$[f(X)]^{m \cdot m'} = [f(X)^m]^{m'} \,(mod\, X^r - 1, p).$$

m' ist auch introspektiv bezüglich $f(X^m)$, das heißt es gilt auch:

32Achtung: Hier steht $(\cdot, \cdot)$ einfach nur für ein Paar von Zahlen und nicht für den größten gemeinsamen Teiler.

$$[f(X^m)]^{m'} = f(X^{m \cdot m'}) \, (mod \, X^{m \cdot r} - 1, p).$$

Es gilt auch $X^{m \cdot r} - 1 = (X^r)^m - (1)^m \underbrace{=}_{(2.5)} (X^r - 1) \sum_{k=0}^{m-1} X^{r \cdot k}$.

$\Rightarrow X^r - 1 \mid X^{mr} - 1$. Somit gilt:

$$[f(X^m)]^{m'} = f(X^{m \cdot m'})(mod X^r - 1, p).$$

Daraus folgt:

$$[f(X)]^{m \cdot m'} = f(X^{m \cdot m'})(mod X^r - 1, p).$$

Das heißt wenn eine Zahl m bezüglich eines Polynoms $f(X)$ introspektiv ist, ist auch $m \cdot m'$ bezüglich des selben Polynoms $f(X)$ introspektiv. □

Lemma 3.3. *wenn m bezüglich der Polynome $f(X)$ und $g(X)$ introspektiv ist. Dann ist m auch bezüglich $f(X) \cdot g(X)$ introspektiv.*

Beweis.

$$[f(X) \cdot g(X)]^m = [f(X)]^m \cdot [g(X)]^m \underbrace{=}_{(3.2)} f(X^m) \cdot g(X^m)(mod \, X^r - 1, p).$$

□

Da n eine Zahl für die

$$(X + a)^n = X^n + a \, (mod \, X^r - 1, n), 0 \leq a \leq \ell$$

gilt, kann man auch folgende Kongruenzen folgern:

$\forall a$, $0 \leq a \leq \ell$.

Bemerkung: Für $a = 0 \Rightarrow (X + 0)^n = X^n + 0$ und damit ist die Gleichung trivialerweise wahr.

Aus der oberen Gleichung kann man folgende Kongruenzen folgern:

$$(X + a)^n = X^n + a \, (\, mod \, X^r - 1, p). \; \forall \, a, \; 0 \leq a \leq \ell. \tag{7}$$

Da p ein Primfaktor von n, gilt auch nach Lemma 3.1

$$(X + a)^p = X^p + a \, (\, mod \, X^r - 1, p). \; \forall \, a \, 0 \leq a, \leq \ell. \tag{8}$$

Wir zeigen nun, dass diese Eigenschaft auch für $\frac{n}{p}$ gilt.

Proposition 3.1. $(X+a)^{\frac{n}{p}} = X^{\frac{n}{p}} + a \,(\, mod\, X^r - 1, p).\ \forall\, a,\ 0 \leq a \leq \ell.$

Beweis.

$$X^n + a \underbrace{=}_{7} (X+a)^n = ((X+a)^p)^{\frac{n}{p}} \underbrace{=}_{8} ((X^p + a))^{\frac{n}{p}} (\, mod\, X^r - 1, p).\ \forall\, a,\ 0 \leq a \leq \ell.$$

$$\Rightarrow\ X^{\frac{n}{p}} + a = (X+a)^{\frac{n}{p}} (\, mod\, X^r - 1, p).\ \forall\, a,\ 0 \leq a \leq \ell. \qquad \square$$

Nun Kann man die Bedingungen für r und a definieren.

3.2.2 AKS-Bedingungen

In diesem Abschnitt werden die erforderlichen Einschränkungen für r und a hergeleitet. Mit Hilfe dieser Einschränkungen wird das sogenannte AKS-Theorem formuliert. Auf Dieses Theorem baut der AKS-Primzahltest auf.

Sei n eine zusammengesetzte Zahl mit dem Primteiler p.

Zuerst werden folgende Mengen definiert:

$$\mathcal{I} := \{\frac{n^i}{p^i} \cdot p^j \mid i, j \geq 0\}.$$

$$\mathcal{P} := \{\prod_{a=0}^{\ell} (X+a)^{e_a} \mid e_a \geq 0\}.$$

Aus Lemma 3.2 und Lemma 3.3 folgt, dass jedes Element $i \in \mathcal{I}$ bezüglich jedes Polynoms $p \in \mathcal{P}$ introspektiv ist.

Allerdings ist es noch nicht klar, was diese Mengen mit der Primalität von n zu tun haben. Aber Es werden endliche Teilmengen der beiden Mengen analysiert, um Bedingungen für r, a zu formulieren und daraus eine Beziehung zwischen n und p zu finden.

Jetzt, dass wir die obigen Mengen definiert haben, ist es möglich Bedingungen für n, r, a und p zu formulieren. Zunächst wird mit den bekannten Größen(Bedingungen) für n, r, a und p angefangen:[33]

$$n \geq 2 \tag{9}$$

$$r \geq 2 \tag{10}$$

33Dabei ist ℓ die polynomielle Obere Schranke für a

$$\ell \geq a > 1 \tag{11}$$

$$p \mid n, p \in \mathbb{P} \tag{12}$$

Die erste endliche Teilmenge von $\mathcal{I}$ ist Menge mit $i, j < k$ für ein $k \in \mathbb{N}$.

Sei $\mathcal{I}_k = \{p^i \cdot (\frac{n}{p})^j \mid 0 \leq i, j < k\} \subseteq \mathcal{I}$.

Theorem 3.4. *Wenn n keine Potenz von p ist, dann sind alle Elemente in $\mathcal{I}_k$ verschieden, das heißt*

$$|\mathcal{I}_k| = k^2.^{34}$$

Für die zweite Menge($\mathcal{P}$) kann man auch eine ähnliche Einschränkung machen:

$$\mathcal{P}_d = \{f \in \mathcal{P} \mid degf < d\} \subseteq \mathcal{P}.$$

Wir interessieren uns im Moment nur für eine obere Schranke für den Grad der Polynome $\in \mathcal{P}_d$, sodass $d \leq \ell$. Dazu betrachtet man die Menge $\mathcal{P}'$, wo jedes Polynom $X + a$ höchstens einmal vorkommt. Da jedes Polynom entweder anwesend ist oder nicht, existieren $2^d - 1$ verschiedene Polynome in $\mathcal{P}'$(ohne das Nullpolynom). Mit dem Nullpolynom erhält man $|\mathcal{P}'| = 2^d.^{35}$

$\mathcal{P}'$ ist aber nach Definition eine Teilmenge von $\mathcal{P}$, daher gilt:

$$|\mathcal{P}| \geq |\mathcal{P}'| = 2^d$$

Daraus kann man folgendes Lemma aufstellen

Lemma 3.5. *Für die Teilmenge $\mathcal{P}_d$ gilt:*

1. $d \leq \ell$.
2. $|\mathcal{P}| \geq 2^d$.

Dies wird später für einen bestimmen Wert von t für den Korrektheitsbeweis benutzt.

Des Weiteren wird die Menge $\mathcal{I}$ anders beschränkt, diesmal aber modulo r. Das heißt:

$$\mathcal{J} = \{x \, mod \, r \mid x \in \mathcal{I}\}$$

34Da $i \neq j$, $\forall i, j < k$(es gibt für jeweils i, j k mögliche Werte).

35Das Nullpolynom ist in der Menge, da für das Nullpolynom ist die vorher erwähnte Kongruenz trivialerweise wahr.

Mit $t = |\mathcal{J}|$.

Nun wird auch eine zweite Teilmenge von $\mathcal{P}$ definiert, dies ist die Teilmenge von $\mathcal{P}$, die in einem endlichen Körper lebt.

Sei $\mathbb{Z}_p$ der Ring der ganzen Zahlen modulo p. Da p eine Primzahl ist(p ist ein Primfaktor von n), ist $\mathbb{Z}_p$ in dem Fall sogar ein endlicher Körper $\mathbb{K}_p$(Def. 2.26) der Ordnung p. Betrachte den Polynomring $\mathbb{K}_p[X]$ (Def. 2.22). Für ein Polynom $q(X) \in \mathbb{K}_p[X]$ ist $\mathbb{K}_p[X]/q(X)$ ein Körper, wenn $q(X)$ irreduzibel in $\mathbb{K}_p[X]$ ist(Theorem 2.17).

Das heißt, es wird ein irreduzibeles Polynom in $\mathbb{K}_p[X]/q(X)$, das $X^r - 1$ teilt, gesucht. Aus der Algebra ist aber bekannt, dass für das Polynom $X^r - 1$ Teiler des r-ten zyklotomischen Polynoms $\Phi_r(X)$ diese Eigenschaft haben.

Lemma 3.6. ***Irreduzible Teiler zyklotomischer Polynome***

Sei $\mathbb{K}_p$ ein Körper und $\Phi_r(X)$ das r-te zyklotomische Polynom über $\mathbb{K}_p$. Dann existiert ein irreduzibler Teiler $h(X)$ von $X^r - 1$ über $\mathbb{K}_p$ vom Grad $o_r(p)$.

Beweis. Sei p eine Primzahl und $\mathbb{K}_p$ der dazugehörige Restklassenkörper. Zudem seien r und p teilerfremd. Nach Def. 2.29 gilt:

$$\Phi_r(X) = \prod_{\substack{1 \le k \le n \\ (k,n)=1}} \left(X - \zeta^k\right).$$

Nach Lemma 2.21 gilt $\Phi_r(X) \mid X^r - 1$. Sei ζ eine beliebige Einheitswurzel von $\Phi_r(X)$ über $\mathbb{K}_p$. $h(X) = \prod_{k=1}^{q}(X - \zeta^{p^k})$ ist genau dann das minimale Polynom von ζ über $\mathbb{K}_p$, wenn q die kleinste positive natürliche Zahl, sodass $\zeta^{p^q} = 1$ beziehungsweise $\zeta^{p^q \bmod r} = 1$(Da ζ eine primitive Einheitswurzel von r ist). Dies ist äquivalent zu $q = o_r(p)$, es ist außerdem offensichtlich, dass diese Relation nur von r, p abhängig ist, also unabhängig von der Wahl von ζ. Das Minimalpolynom $h(X)$ teilt jedes Polynom mit den Nullstellen ζ. Daher ist $h(X)$ irreduzibl über $\mathbb{K}_p$ und teilt $\Phi_r(X) \Rightarrow h(X) \mid X^r - 1$, folglich zerfällt $\Phi_r(X)$ in Polynome über $\mathbb{K}_p$ jeweils vom Grad $o_r(p)$. Das bedeutet $\Phi_r(X)$ zerfällt in $\frac{deg(\Phi_r(X))}{o_r(p)} = \frac{\phi(p)}{o_r(p)}$ irreduzible Polynome vom Grad $o_r(p)$ über $\mathbb{K}_p$. Das heißt es existiert ein über $\mathbb{K}_p$ irreduzibler Teiler von $X^r - 1$ vom Grad $o_r(p)$. □

Jetzt kann man fordern, dass $(r, p) = 1$. Sei $h(X)$ einer der irreduzibelen Faktoren von $\Phi_r X)$ in $\mathbb{K}_p$, dann erhält man den Körper $\mathbb{K}_p/h(X)$.

Nun kann die zweite Teilmenge von $\mathcal{P}$ definiert werden(Diese Menge ist sogar eine Gruppe).

Die zweite Menge $\mathcal{G}$ sei die multiplikative Gruppe aller Restklassen von Polynomen in $\mathcal{P}$ modulo $h(X)$ und p, das bedeutet $\mathcal{G} = \{f \ (mod\, h(X), p) \mid f \in \mathcal{P}\}$.

Jedes Element von $\mathcal{P}$ kann auf Elemente in $\mathcal{G}$ abgebildet werden. Interessant sind in diesem Fall nur Elemente von $\mathcal{P}$, die auf verschiedene Elemente in $\mathcal{G}$ abgebildet werden,

da dies uns ermöglicht, eine untere Schranke von $\mathcal{G}$ zu finden. Eine einfache untere Schranke erhält man, wenn alle $X + a$ in $\mathcal{G}$ verschieden sind. Dies ist der Fall, wenn $deg(h(X)) > 1$ und $p \leq \ell$, wobei $\ell \geq a$.

Man kann auch zeigen, dass verschiedene Elemente von $\mathcal{P}_d$ auf verschiedene Elemente in $\mathcal{G}$ abgebildet werden, falls $d \leq t$. Seien $f(X), g(X) \in \mathcal{P}_d$, mit $f(X) = g(X) \,(mod\, h(X), p)$. Für $m \in \mathcal{I}$ gilt:

$$f(X^m) = [f(X)]^m \,(mod\, X^r - 1, p)$$

und

$$g(X^m) = [g(X)]^m \,(mod\, X^r - 1, p)$$

Da $f(X) = g(X) \,(mod\, h(X), p$ gilt auch

$$f(X^m) = g(X^m) \,(mod\, X^r - 1, p)$$

Da $h(X) \mid X^r - 1$ gilt

$$f(X^m) = g(X^m) \,(mod\, h(X), p)$$

Folglich sind alle $X^m, \forall\, m \in \mathcal{I}$ die Wurzeln des Polynoms $Q(Y) = f(Y) - g(X)$ in $\mathbb{K}_p/h(X)$, insbesondere alle $X^m, \forall\, m \in \mathcal{J}$. Nach Theorem 2.11 folgt $deg\, Q(Y) > t$.[36] Es gilt außerdem, dass $deg\, Q(Y) < d$.[37] Daraus folgt $d \leq t$ und das ist ein Widerspruch zu Theorem 2.11. Deshalb gilt $f(X) \neq g(X) \,(mod\, h(X), p)$. Aber da $f(X), g(X)$ beliebig sind, werden verschiedene Elemente aus $\mathcal{P}_d$ die auf verschiedene Elemente von $\mathcal{G}$ abgebildet, wenn $d \leq t$.

Mit diesen Bedingungen kann man nun eine untere Schranke für $\mathcal{G}$ finden. Für $d \leq t, p \geq \ell$ und $deg(h(X)) \underbrace{=}_{3.6} o_r(n) > 1$ gilt:

$$|\mathcal{G}| \geq |\mathcal{P}_d| \geq 2^d.$$[38]

Diese Bedingungen können in einem Lemma zusammengefasst werden.

Lemma 3.7. *Sei n eine Potenz von p. Des Weiteren seien $\mathcal{P}_d = \{f \in \mathcal{P} \mid degf < d\}$, $\mathcal{J} = \{x \,mod\, r \mid x \in I\}, \mathcal{G} = \{f \,(mod\, h(X), p) \mid f \in \mathcal{P}\}$, mit $|\mathcal{J}| = t \geq d = |\mathcal{P}_d|, o_r(p) > 1$ und $\ell = max\{a \mid (a, n) = 1\} \geq d$, dann gilt:*

1. $|\mathcal{G}| > 2^d$.
2. $p \geq \ell$.

36Die Wurzel von $Q(Y)$ sind X^m und es gibt $|\mathcal{J}| = t$ solche Wurzeln, da $m \in \mathcal{J}$.
37Dies folgt einfach aus der Definition der Menge $\mathcal{P}_d$.

Wir nehmen nun an, dass n keine Potenz von p ist. Des Weiteren seien $|\mathcal{J}| = t$(wie vorher) und $s > \sqrt{t}$, dann gilt $|\mathcal{I}_s| = s^2$.[39]

Nach Theorem 2.6 gilt, dass es zwei Elemente $m_1, m_2 \in \mathcal{I}_s$ existieren, die auf das selbe Element in $\mathcal{J}$ abgebildet werden(da $s^2 > t$). Das heißt es gilt:

$$m_1 = m_2 \, (mod r).$$

Deswegen gilt für ein Polynom $g(X) \in \mathcal{P}$ das folgende:

$$[g(X)]^{m_1} = [g(X)]^{m_2} \, (mod \, h(X), p).$$

Das bedeutet, dass $g(X) \in \mathcal{G}$ eine Wurzel von $Q(Y) = Y^{m_1} - Y^{m_2}$ in $\mathbb{K}_p/h(X)$ ist. $g(X) \in \mathcal{G}$ ist aber beliebig und daher gilt, dass $Q(Y)$ mindestens $|\mathcal{G}|$ Wurzeln hat.

Außerdem ist $deg(Q(Y)) = max\{m_1, m_2\} \leq p^{s-1} \cdot (\frac{n}{p})^{s-1} = n^{n-1}$. Nach Theorem 2.11 gilt auch $|\mathcal{G}| \leq n^{s-1}$.

Das bedeutet, wenn n keine Potenz von p ist, gilt $|\mathcal{G}| \leq n^{s-1}$ und wenn n eine Potenz ist, gilt $|\mathcal{G}| > n^{n-1}$.

Nach Theorem 3.7 gilt $|\mathcal{G}| \geq 2^d, d \leq t$. Das heißt mit einer geeigneten Wahl von d, s, sodass $2^d > n^{s-1}$, dann folgt $n = p^m$.[40] Zuerst kann man log von $2^d > n^{s-1}$ nehmen. Daraus folgt $d > (s-1) \cdot log\, n$. Da $d \leq t$, gilt auch, dass $t > d > (s-1) \cdot log\, n$. Aus dieser Ungleichung folgt $s < \frac{t}{log\, n} + 1$. s ist nach Voraussetzung größer als $\sqrt{t}$, folglich $\sqrt{t} < \frac{t}{log\, n} + 1$. Nach Umstellung gilt $log\, n < \frac{t}{\sqrt{t}-1}$. Außerdem gilt auch $\sqrt{t} < \frac{t}{\sqrt{t}-1}$[41] und daher ist es ausreichend zu fordern, dass $t > log^2 n$[42], damit n eine Potenz sein kann. s ist eine natürliche Zahl, das heißt $s = \lfloor t \rfloor + 1$[43], folglich gilt:

$$d > (s-1) \cdot log\, n = (\sqrt{t} + 1 - 1) \cdot log\, n = \sqrt{t} \cdot log\, n.$$

Das Ziel ist eine Primzahlpotenz zu erzeugen. n ist eine Primzahlpotenz, wenn $|\mathcal{G}| > n^{s-1}$. Alle bisherige Bedingungen können ausschließlich mit n, r ausgedrückt werden. Dies sind:

$$\ell > d > \sqrt{t} \cdot log\, n. \;\; (Lemma\, 3.5).$$

und

$$t > log^2 n.$$

39 Zur Erinnerung: $\mathcal{I}_k = \{p^i \cdot (\frac{n}{p})^j \mid 0 \leq i, j < k\} \subseteq \mathcal{I}$.
40 $2^d > n^{s-1} \Rightarrow |\mathcal{G}| > n^{s-1} \Rightarrow n = p^m, m \in \mathbb{N}$[Aka12].
41 $\frac{t}{\sqrt{t}-1} = \frac{\sqrt{t} \cdot \sqrt{t}}{\sqrt{t}-1} > \frac{(\sqrt{t}-1) \cdot \sqrt{t}}{\sqrt{t}-1} > \sqrt{t}$.
42 $t > log\, n \cdot (\sqrt{t} - 1) > log\, n \cdot \sqrt{t} \Rightarrow t > log^2 n$.
43 $\sqrt{t}$ ist nicht zwangsweise eine natürliche Zahl.

Die Menge $\mathcal{J} = \{x \, mod \, r \mid x \in \mathcal{I}\}$, mit $|\mathcal{J}| = t$ wird von n und $\frac{n}{p}$ erzeugt. Daher gilt $t > o_r(p)$, und $o_r(p) \geq o_p(n)$(da $p \mid n$). Nun können die Bedingungen $t > log^2 n$ und $o_r(p) > 1$ durch eine Bedingung $o_r(n) > log^2 n$ ersetzt werden. Zudem gilt $(r, n) = 1$, aber da $\mathcal{J} \subseteq \mathbb{Z}_r^*$ gilt auch:

$$|\mathbb{Z}_r^*| = \phi(r) > |\mathcal{J}| = t.$$

Daraus folgt $\ell \geq d > \sqrt{\phi(r)} log \, n$, da $\phi(r)$ das Maximum von t ist.

Schließlich kann man diese Resultate in einem Satz zusammenfassen. Das Ziel war Bedingungen für n, r, a und p zu finden, sodass n eine Potenz von p ist. Es wurde gezeigt, dass folgende drei Bedingungen dafür erforderlich sind:

1. $n \geq 2$ keine Primfaktoren $< \ell$ hat.
2. $o_r(n) > log^2 n$.
3. $\ell > \lfloor \sqrt{\phi(r)} \cdot log \, n \rfloor$.

Diese Bedingungen können in einem Theorem zusammengefasst werden.

Theorem 3.8 (**AKS-Theorem**). *Sei $n \geq 2$ mit $(n, r) = 1, o_r(n) > log^2 n, \ell > \lfloor \sqrt{\phi(r)} log \rfloor$ und n hat keine Primfaktoren kleiner ℓ. Außerdem sei*

$$(X + a)^n = X^n + a \, (mod \, X^r - 1, n)$$

Für alle $0 \leq a \leq \ell$ erfüllt, dann ist n eine Potenz von p.

Mithilfe von Theorem 3.8 kann jetzt ein Algorithmus entwickelt werden, sodass für ein geeignetes r ($o_r(n) > log^2 n$) nur $\lfloor \phi(r) log \, n \rfloor$ verschiedene a Werte im Polynom getestet werden, da Zahlen mit einem Primfaktor $> \lfloor \phi(r) log \, n \rfloor$ Primzahlpotenzen sind. Das heißt, dass die Anzahl an zu testenden a Werte von einem Polynom von $log \, n$ abhängig ist. Es bleibt nur zu zeigen, dass das r auch von einem Polynom von $log \, n$ abhängig ist und dies wird im vierten Kapitel gezeigt.

3.3 Der Algorithmus

Notation:

- $o_r(n)$ bezeichnet die multiplikative Ordnung der Zahl n mod r (Def. 2.6).
- (a, b) ist der größte gemeinsame Teiler (Def. 2.1).
- ϕ ist die eulersche Phi-Funktion (Def. 2.7).
- COMPOSITE: die Zahl n ist zusammengesetzt($n \notin \mathbb{P}$).
- PRIME: die Zahl n ist prim ($n \in \mathbb{P}$).

Algorithm 1 AKS-Primzahltest

Input: $n \in \mathbb{N}, n \geq 2$.

1. **if** $n = a^b, a \in \mathbb{N}, b \geq 1$, **return** COMPOSITE.
2. finde das kleinste r, sodass $o_r(n) > log^2 n$.
3. **if** $1 < (a, n) < n, a \geq n$, **return** COMPOSITE.
4. **if** $n \leq r$, **return** PRIME.
5. **for** a = 1 to $\lfloor \sqrt{\phi(r)} log(n) \rfloor$:
   if $(X + a)^n \neq X^n + a (\text{mod } X^r - 1, n)$, **return** COMPOSITE.
6. **return** PRIME.

Beschreibung des Algorithmus:

Schritt 1: Im ersten Schritt sucht der Algorithmus zwei Zahlen $a, b \in \mathbb{N}$, $b > 1$, sodass $n = a^b$. Wenn solche Zahlen existieren, gibt er COMPOSITE zurück.[44]

Schritt 2: Hier probiert der Algorithmus sukzessive Werte von r aus, bis $n^k \neq 1 (mod\, r)$, $\forall k \leq log^2 n$. Hier gibt es zwei Fälle. $o_r(n) > log^2 n$ oder $(r, n) > 1$, im ersten Fall ist das kleinste k, für es $n^k = 1 (mod\, r)$ erfüllt $> log^2 n$. Während im zweiten Fall kein k existiert, für es $n^k = 1 (mod\, r)$ erfüllt ist.

Schritt 3: In diesem Schritt wird überprüft, ob n und a, $a \leq r$ gemeinsame Faktoren haben.

Schritt 4: Wenn keine Primfaktoren bei Schritt 3 gefunden worden sind, gibt der Algorithmus bei diesem Schritt PRIME zurück. Da $n \leq r$ und $(k, n) = 1$, $\forall k \leq r$(sonst hätte er beim 3. Schritt schon COMPOSITE zurückgegeben).

Schritt 5: In diesem Schritt überprüft der Algorithmus, ob die Gleichung $(X + a)^n = X^n + a (mod\, X^r - 1, n)$ für alle a Werte erfüllt ist.

Schritt 6: Wenn kein a in Schritt 5 gefunden wurde, sodass die Gleichung nicht erfüllt ist. Dann muss die Zahl prim sein.

44 Primzahlen lassen sich nicht als eine Potenz von einer anderen Zahl darstellen.

3.4 Korrektheitsbeweis

In diesem Abschnitt wird der Korrektheitsbeweis des AKS-Algorithmus durchgeführt, um zu zeigen, dass der AKS-Algorithmus korrekt ist, und als ein Primzahltest anwendbar ist.

Theorem 3.9. ***Hauptsatz der Korrektheit***

Der Algorithmus gibt genau dann PRIME zurück, wenn n eine Primzahl ist.

Dieses Ergebnis wird durch eine Reihe von Lemmata festgestellt. Der Korrektheitsbeweis lässt sich in zwei Teile zerlegen, der erste Teil befasst sich mit der Hinrichtung des Beweises. Also, dass der Algorithmus PRIME liefert, wenn die Eingabe eine Primzahl ist. Dies ist aber trivial und braucht keine Lemmata. Für den zweiten Teil(Rückrichtung) sind jedoch mehrere Lemmata und Sätze nötig.

3.4.1 Erster Teil des Beweises($\Rightarrow$)

Theorem 3.10. *Wenn n eine Primzahl ist, dann gibt der Algorithmus PRIME zurück.*

Beweis. Wenn n eine Primzahl ist, gibt der erste Schritt (in Algorithmus 3.3) niemals COMPOSITE zurück. Da keine Primzahl sich als $a^b, b > 1$ darstellen lässt, sonst wäre n durch a teilbar. Der dritte Schritt kann auch niemals COMPOSITE zurückgeben, da $(a, n) = 1, \forall a$. Nach Lemma (3.1) kann Schritt 5 auch nie COMPOSITE zurückgeben. Daher muss der Algorithmus entweder bei viertem oder bei sechstem Schritt PRIME zurückgeben. □

3.4.2 Zweiter Teil des Beweises($\Leftarrow$)

Hier wird die Rückrichtung behandelt, das heißt wenn die Ausgabe PRIME ist, dann ist n eine Primzahl. Wir werden uns nur auf die Schritte konzentrieren, die PRIME zurückgeben. Dies sind die Schritte 4 und 6.

Wenn der Algorithmus bei Schritt 4 PRIME zurückgibt, dann muss n eine Primzahl sein, da sonst der Algorithmus einen nicht trivialen Faktor von n in Schritt 3 gefunden hätte. Das heißt, es bleibt nur der Fall bei Schritt 6. Der Algorithmus hat Zwei Hauptschritte, das sind Schritt 2 und Schritt 5. Bei Schritt 2 wird ein geeignetes r gefunden und bei Schritt 5 wird verifiziert, ob die Gleichung

$$(X + a)^n = X^n + a \,(mod\, X^r - 1, n)$$

für mehrere a Werte gilt.

Daher wird im folgenden nur Schritt 5 analysieren, hierzu wird zunächst folgende Proposition betrachtet:

Proposition 3.2. *Im fünften Schritt des Algorithmus ist $p > r$.*

Beweis. Sei $p \leq r$, wenn $p < r$ ist, dann gibt der Algorithmus im dritten Schritt COMPOSITE zurück. Falls $p = n$ ist, dann gibt der Algorithmus im vierten Schritt PRIME zurück. Das bedeutet, wenn der Algorithmus Schritt 5 erreicht hat, dann ist $p > r$. □

Proposition 3.3. *wenn $o_r(n) > 1$, dann existiert ein Primfaktor p von n, sodass $o_r(p) > 1$.*

Beweis. Sei $o_r(n) > 1$, mit der Primfaktorzerlegung

$$n = \prod_{i=1}^{M} p_i^{e_i}$$

Nun wird angenommen, dass $o_r(p_i) = 1$, $\forall 1 \leq i \leq M$ (es existiert kein $p_i : o_r(p_i) > 1$).

$\Rightarrow p_i = 1 (\text{mod } r)$, $\forall i$. Folglich gilt

$$n = \prod_{i=1}^{M} p_i^{e_i} \,(mod\, r) = \prod_{i=1}^{M} (1)^{e_i} \,(mod\, r) = 1 \,(mod\, r)$$

Daraus wird ersichtlich, dass $o_r(n) = 1$ und das ist ein Widerspruch zur Voraussetzung, dass $o_r(n) > 1$. Daher existiert mindestens ein Primfaktor $p : o_r(p) > 1$. □

Nach Proposition 3.3 muss für $o_r(n) > 1$ ein Primteiler p von n(Aus der Primfaktorzerlegung) existieren, sodass $o_r(p) > 1$. Seien $p, n \in \mathbb{Z}_r^*$ fest. Ferner sei $\ell = \lfloor \sqrt{\phi(r)} log n \rfloor$.

Beobachtung: $r < n$, da sonst der Algorithmus im vierten Schritt PRIME zurückgegeben hätte(der Algorithmus hätte Schritt 5 nicht erreicht).

Der 5. Schritt des Algorithmus überprüft ℓ Gleichungen. Nach Voraussetzung ist die Ausgabe PRIME . Das heißt, für alle $\ell + 1$ Gleichungen(inkl. $a = 0$) gilt:

Hier werden die Mengen $\mathcal{I}$ und $\mathcal{P}$ aus dem letzten Abschnitt wieder gebraucht.

Zu Erinnerung: $\mathcal{I} := \{p^i \cdot (\frac{n}{p})^j \mid i, j > 0\}$ und $\mathcal{P} := \{\prod_{a=0}^{\ell} (X + a)^{e_a} \mid e_a \geq 0\}$. Aufbauend auf diese Mengen werden zwei Gruppen definiert, die für den Beweis von Bedeutung sind. Das Ziel ist eine Abschätzung für diese Gruppen zu machen, und dann dadurch einen Widerspruch herzuleiten.

Beide Gruppen wurden schon im vorherigen Abschnitt definiert, aber sie werden hier der Vollständigkeit halber nochmal definiert und es werden darüber hinaus Eigenschaften für diese Gruppen explizit definiert.

Definition 3.3. Die multiplikative Gruppe $\mathcal{J} \subseteq \mathbb{Z}_n^*$

Die erste Gruppe $\mathcal{J}$ sei die Gruppe aller Restklassen in $\mathcal{I}$ modulo r, das heißt $\mathcal{J} = \{(\frac{n}{p})^i \cdot p^j (mod\, r) \mid i, j \geq 0\}$. Da $(n, r) = (p, r) = 1$ ist $\mathcal{J}$ eine multiplikative Untergruppe von $\mathbb{Z}_r^*$.[45] Sei die Kardinalität im weiteren $|\mathcal{J}| = t$. $\mathcal{J}$ wird von n und p modulo r erzeugt, und da $o_r(n) > log^2 n$, gilt auch $t > log^2 n$.[46]

Definition 3.4. Die multiplikative Gruppe $\mathcal{G}$

Die zweite Gruppe $\mathcal{G}$ sei die multiplikative Gruppe aller Restklassen von Polynomen aus der Menge $\mathcal{P}$ modulo $h(X)$ und p, wobei $h(X)$ ein irreduzibler Teiler des r-ten zyklotomischen Polynoms $\Phi_r(X)$ über $\mathbb{K}_p/h(X)$ ist. Nach Lemma 3.6 $deg\, h(X) = o_r(p)$. Das heißt $\mathcal{G} = \{f\,(mod\, X^r - 1, p) \mid f \in \mathcal{P}\}$. $\mathcal{G}$ ist eine Untergruppe von der multiplikativen Gruppe von $\mathbb{K} = \mathbb{K}_p[X]/h(X)$[47] und wird von $X, X+1, X+2, ..., X+\ell$ im Körper $\mathbb{K}$ erzeugt

Nun kann eine untere Schranke für $\mathcal{G}$ bestimmt werden.[48]

Lemma 3.11. ***(Untere Schranke für $|\mathcal{G}|$).***

$|\mathcal{G}| \geq \binom{t+\ell}{t-1}$

Beweis. Zunächst sei $h(X)$ ein irreduzibeler Faktor(in $\mathbb{K}$) des r-ten zyklotomischen Polynoms $\Phi_r(X)$ und somit auch von $X^r - 1$.[49] Das heißt $X^r - 1 = A(X) \cdot \Phi_r(X) = A(X) \cdot B(X) \cdot h(X)$ mit $A(X), B(X) \in \mathbb{K}_p[X]$. Daraus folgt, dass für eine Wurzel/Nullstelle α des Polynoms $h(X)$ stets $\alpha^r = 1$ gilt. folglich ist X eine primitive r-te Einheitswurzel in $\mathbb{K}$.

Zuerst wird gezeigt, dass alle $f(X), g(X) \in \mathcal{P}$, mit $f(X) \neq g(X)$ vom Grad $< t$ auf zwei verschiedene Elemente in $\mathcal{G}$ abgebildet werden.

Angenommen $f(X) = g(X)$ in $\mathbb{K}$. Weiterhin seien $m \in \mathcal{I}$ beliebig und $[f(X)]^m = [g(X)]^m$ in $\mathbb{K}$. m ist introspektiv bezüglich $f(X), g(X)$ daher gilt:

$$f(X^m) = g(X^m)(mod\, X^r - 1, p).$$

Da $h(X) \mid X^r - 1$ gilt auch:

$$f(X^m) = g(X^m)(mod\, h(X), p).$$

folglich gilt $f(X^m) = g(X^m)$ in $\mathbb{K}$. Daraus folgt, dass X^m eine Wurzel des Polynoms $Q(Y) = f(Y) - g(Y)$, $\forall m \in \mathcal{I}$ ist. Da X eine primitive r-te Einheitswurzel von $\Phi_r(X)$ in

45 $Z_r^* = \{x \in Z_r \mid (x, r) = 1\}$.
46 Da $n \in \mathcal{J}$ mit Ordnung $> log^2$, folgt $t = |\mathcal{J}| > log^2 n$.
47 $\mathcal{G}$ ist offensichtlich unter Multiplikation abgeschlossen und enthält ein Identität-Element
48 Die untere Schranke beschreibt in diesem Fall die Anzahl der eindeutigen Polynome in der Gruppe $\mathcal{G}$.
49 Dies folgt aus Theorem 2.21

$\mathbb{K}$ und $(m,r) = 1(\mathcal{I} \subseteq \mathbb{Z}_r^*)$, $\forall m \in \mathcal{J}$, folgt auch, dass X^m eine r-te primitive Einheitswurzel in $\mathbb{K}$ ist.

Es existieren also $|\mathcal{J}| = t$ verschiedene Wurzeln von Q in $\mathbb{K}$. Nach der Wahl von f, g ist aber $deg\, Q < t$. Das ist ein Widerspruch zu Theorem 2.11. Daraus folgt, dass $f(X) \neq g(X)$ in $\mathbb{K}$.

Es gilt auch, dass $i \neq j \in \mathbb{K}_p, \forall\, 1 \leq i \neq j \leq \ell = \lfloor\sqrt{\phi(r)} log n\rfloor$(siehe Theorem 3.8). Da $o_r(n) > log^2 n$, gilt auch $r > log^2 n$ und folglich gilt:[50]

$$\ell = \lfloor\sqrt{\phi(r)}\, log\, n\rfloor < \sqrt{r}\, log\, n < r \underbrace{\leq}_{(3.2)} p$$

.

Daher sind $X, X+1, ..., X+\ell$ verschieden in $\mathbb{K}$. Außerdem gilt $deg\, h(X) > 1$ und $X + a \neq 0$ in $\mathbb{K}$, $\forall a$, $0 \leq a \leq \ell$. Das bedeutet es existieren $\ell + 1$ verschiedene Polynome vom Grad 1 (Da $deg\, X + a = 1$). Nun zählen wir alle eindeutige Polynome vom Grad $< t$, das heißt wir suchen die Polynome der Form $X^{e_0}\,(X+1)^{e_1} \cdots (X+\ell)^{e_\ell}$, wobei $\sum_{i=0}^{\ell} e_i < t$. Die Anzahl der Polynome mit Grad $< t$ in $\mathcal{G}$ erhält man durch die Kombination mit Wiederholung. Die Formel für Kombination mit Wiederholung für n aus k ist $\binom{n+k-1}{k}$. In diesem Fall ist $n = t$ und $k = \ell + 1$. Das heißt es existieren mindestens

$$\binom{t+\ell}{\ell+1} = {}^{51}\binom{t+\ell}{(t+\ell)-(\ell+1)} = \binom{t+\ell}{t-1}$$

Polynome vom Grad $< t$ in $\mathcal{G}$. Daraus folgt $|\mathcal{G}| \geq \binom{t+\ell}{t-1}$ □

Wenn n keine Potenz von p ist, erhält man auch eine obere Schranke für $|\mathcal{G}|$. Mit dieser oberen Schranke kann nachher einen Widerspruch konstruieren werden.

Lemma 3.12. ***(Obere Schranke für $|\mathcal{G}|$).***

Wenn n keine Potenz von p, dann gilt:

$$|\mathcal{G}| \leq n^{\sqrt{t}} \tag{13}$$

Beweis. Zuerst wird die Menge $\mathcal{I}_k$ aus Abschnitt 3.2 mit $k = \sqrt{t}$ betrachtet:

$$\mathcal{I}_{\sqrt{t}} = \{(\frac{n}{p})^i p^j \mid 0 \leq i, j \leq \sqrt{t}\} \subseteq \mathcal{I}. \tag{14}$$

Wenn n keine Potenz von p ist, sind alle p^j und $(\frac{n}{p})^i$ $\forall\, 1 < i \neq j < \sqrt{t}$ verschieden. Das heißt es gilt $|\mathcal{I}_{\sqrt{t}}| = (\lfloor\sqrt{t}\rfloor + 1)^2 > t = |\mathcal{J}|$[52]. Deshalb müssen (nach Theorem 2.6)

50 r ist (nach Voraussetzung)eine Primzahl und daher gilt: $\phi(r) = r - 1 \Rightarrow \phi(r) < r$. Außerdem gilt auch: $r > o_r(n) > log^2 n \Rightarrow \sqrt{r} > log\, n$

51 Dies folgt aus der Tatsache, dass $\binom{n}{k} = \binom{n}{n-k}$.

52 Da es $\lfloor\sqrt{t}\rfloor + 1$ mögliche Werte für jeweils i und j gibt

mindestens zwei Elemente $m_1, m_2 \in \mathcal{I}_{\sqrt{t}}$ existieren, sodass $m_1 = m_2 \, mod \, r$. O.B.d.A sei $m_1 > m_2$, dann gilt:

$$X^{m_1} = X^{m_2} \, (mod \, X^r - 1).$$

Sei nun $f \in \mathcal{P}$, dann gilt:

$$\begin{aligned} f(X)^{m_1} &= f(X^{m_1}) \, (mod \, X^r - 1, p) \\ &= f(X^{m_2}) \, (mod \, X^r - 1, p) \\ &= f(X)^{m_2} \, (mod \, X^r - 1, p). \end{aligned} \tag{15}$$

Aus (15) folgt $[f(X)]^{m_1} = [f(X)]^{m_2}$ in $\mathbb{K}$. Demnach ist $f(X) \in \mathcal{G}$ eine Wurzel des Polynoms $Q^{'}(Y) = Y^{m_1} - Y^{m_2}$ im Körper $\mathbb{K}$. Da $f(X) \in \mathcal{G}$ beliebig ist, gibt es $|\mathcal{G}|$ verschiedene Wurzeln von $Q^{'}(Y)$ im Körper $\mathbb{K}$. Außerdem gilt auch:

$$deg \, Q^{'}(Y) = m_1 \leq (\frac{n}{p} \cdot p)^{\lfloor \sqrt{t} \rfloor} \leq n^{\sqrt{t}}. \tag{16}$$

Nach (2.11) gilt $|\mathcal{G}| \leq deg \, Q^{'}(Y)$, und folglich gilt $|\mathcal{G}| \leq n^{\sqrt{t}}$.

□

Nun kann der Hauptsatz der Korrektheit bewiesen werden. Die Idee ist $\mathcal{G}$ nochmal abzuschätzen und durch diese Abschätzung einen Widerspruch herzuleiten.

Theorem 3.13. *Ist die Ausgabe PRIME, dann ist n eine Primzahl*

Beweis. Nach Lemma 3.11 gilt für $t = |\mathcal{J}|$, und $\ell = \lfloor \sqrt{\phi(r)} \, log \, n \rfloor$ das folgende:

$$\begin{aligned} |\mathcal{G}| &\geq \binom{t+\ell}{t-1} \\ &\geq \binom{\ell + 1 + \lfloor \sqrt{t} \, log \, n \rfloor}{\lfloor \sqrt{t} log \, n \rfloor} && (\, t = \sqrt{t}\,\sqrt{t} \geq \sqrt{t} \, log \, n^{53}) \\ &\geq \binom{2\lfloor \sqrt{t} \, log \, n \rfloor + 1}{\lfloor \sqrt{t} log \, n \rfloor} && (\, \ell = \lfloor \sqrt{\phi(r)} log \, n \rfloor \geq \lfloor \sqrt{t} \, log \, n \rfloor^{54}) \\ &> 2^{\lfloor \sqrt{t} \, log \, n \rfloor + 1} && (\, Theorem \, 2.8\,) \\ &\geq n^{\sqrt{t}}. \end{aligned}$$

Falls n keine Potenz von p ist, gilt Gemäß Lemma 3.12, dass $|\mathcal{G}| \leq n^{\sqrt{t}}$. Daraus kann man folgern, dass $n = p^k$, $k \geq 1$. Aber die Ausgabe ist nach Voraussetzung PRIME[55], folglich gilt $k = 1 \Rightarrow n = p$. Somit ist n eine Primzahl. □

Aus Theorem 3.10 und Theorem 3.13 folgt der Hauptsatz der Korrektheit.

53 $o_r(n) = |\{n^i \, mod \, r \mid i \geq 0\}| > log^2 n$, diese Menge ist offensichtlich eine Teil menge von $\mathcal{J}$. Folglich gilt $t = |\mathcal{J}| > log^2 n \Rightarrow \sqrt{t} > log\, n$.

54 $\mathcal{J}$ wird von $n, p \in \mathbb{Z}_r^*$ erzeugt $\Rightarrow |\mathcal{J}| \leq |\mathbb{Z}_r^*|$, da $|\mathbb{Z}_r^* - \{n, p\}| \geq 0$. Daraus folgt $|\mathcal{J}| = t \leq \phi(n) = |\mathbb{Z}_r^*|$.

55 Die Ausgabe ist Nach Voraussetzung eine Primzahl, und kann daher keine Potenz einer anderen Zahl sein.

4 Laufzeitanalyse

In diesem Kapitel wird die Laufzeit des AKS-Algorithmus untersucht. Aber Bevor man mit der eigentlichen Laufzeitanalyse beginnt, muss man zuerst zeigen, dass das geeignete r mit $o_r(n) > log^2 n$ im zweiten Schritt des Algorithmus existiert, und dass die Obere Schranke für solches r polynomiell von der Eingabegröße $(log\, n)$ abhängig ist.

4.1 Abschätzung von r

In diesem Abschnitt wird der Existenzbeweis für das geeignete r durchgeführt.

Zuerst wird folgendes Lemma über das kleinste gemeinsame Vielfaches von n Zahlen benötigt[56].

Lemma 4.1. *Für $n \geq 7$ gilt:*

$$kgV(n) \geq 2^n. \tag{17}$$

Beweis. Sei $d_n = kgV_{1\leq m\leq n}\{m\}$, zuerst wird gezeigt, dass $m\binom{n}{m} \mid d_n$.

Wir betrachten das folgende Integral für $n \geq 1$:

$$\begin{aligned} I_{n,m} &= \int_0^1 x^{m-1}(1-x)^{n-m}dx \underbrace{=}_{(2.4)} \int_0^1 x^{m-1} \sum_{r=0}^{n-m}(-1)^r \binom{n-m}{r} \cdot x^r dx \\ &= \sum_{r=0}^{n-m}(-1)^r \binom{n-m}{r} \cdot \int_0^1 x^{m+r-1} = \sum_{r=0}^{n-m}(-1)^r \binom{n-m}{r} \cdot \frac{x^{m+r}}{m+r}\Big|_0^1 \\ &= \sum_{r=0}^{n-m}(-1)^r \binom{n-m}{r} \cdot \frac{1}{m+r} \end{aligned} \tag{18}$$

Aus (18) ist es leicht zu sehen, dass $r \leq n-m$ und folglich $r+m \leq n$. Das heißt $m+r \mid d_n$. Dabei ist es offensichtlich, dass $d_n \cdot I_{n,m} \in \mathbb{N}$.

Wir wollen nun durch Induktion nach $n - m = l \in \mathbb{N}$ und partielle Integration zeigen, dass für $n, m \in \mathbb{N}$, $1 \leq m \leq n$

$$I_{n,m} = \int_0^1 x^{m-1}(1-x)^l dx = \frac{1}{m \cdot \binom{n}{m}} \tag{19}$$

immer gilt.

56Der Beweis basiert auf Argumentationen in [Oli09] und [Nai82]

IA: $\forall n, m \in \mathbb{N}$ mit $n - m = 0$ gilt:

$$I_{n,m} = \int_0^1 x^{m-1} \cdot (1-x)^0 dx = \int_0^1 x^{m-1} dx = \left[\frac{x^m}{m}\right]_0^1 = \frac{1}{m} = \frac{1}{m \cdot \binom{m}{m}}$$

IH: Nun wird angenommen, dass $\forall n, m \in \mathbb{N}$ mit $n - m = l$ die Aussage (19) gilt.

IS: zu zeigen: (19) gilt $\forall n, m \in \mathbb{N}$ mit $n - m = l + 1$.

Partielle Integration:

$$I_{n,m} = \int_0^1 \underbrace{x^{m-1}}_{f'} \cdot \underbrace{(1-x)^{n-m}}_{g} dx = \left[\underbrace{\frac{x^m}{m}}_{f} \cdot \underbrace{(1-x)^{n-m+1}}_{g}\right]_0^1 - \int_0^1 \underbrace{\frac{x^m}{m}}_{f} \cdot \underbrace{(-1) \cdot (n-m) \cdot (1-x)^{l+1-1}}_{g'} dx$$

$$= 0 + \int_0^1 \frac{(n-m) \cdot x^m}{m} \cdot (1-x)^l dx$$

$$= \frac{(n-m)}{m} \cdot \int_0^1 x^{(m+1)-1} \cdot (1-x)^l dx$$

$$\underbrace{=}_{IH} \frac{(n-m)}{m} \cdot \frac{1}{(m+1) \cdot \binom{n}{m+1}} = \frac{(n-m)}{m} \cdot \frac{(n-(m+1))! \cdot (m+1)!}{(m+1) \cdot n!}$$

$$= \frac{(n-m)! \cdot m!}{m \cdot n!} = \frac{1}{m \cdot \binom{n}{m}}$$

Somit ist Aussage(19) wahr.

Demzufolge gilt:

$$\frac{d_n}{m \cdot \binom{n}{m}} \in \mathbb{N} \Rightarrow m \cdot \binom{n}{m} \mid d_n \tag{20}$$

$\forall m$ mit $1 \leq m \leq n$.

Somit gilt auch:

$$n\binom{2n}{n} \mid d_{2n} \tag{21}$$

Beziehungsweise

$$(2n+1)\binom{2n}{n} = (n+1)\binom{2n+1}{n+1} \mid d_{2n+1}. \tag{22}$$

$$\Rightarrow n\binom{2n}{n} \mid d_{2n} \mid d_{2n+1}$$

Da n und $2n+1$ teilerfremd sind gilt auch:

$$n \cdot (2n+1) \cdot \binom{2n}{n} \mid d_{2n+1}$$

Daraus folgt:

$$n \cdot (2n+1) \cdot \binom{2n}{n} \leq d_{2n+1} \leq d_{2n+2}$$

Nun zeigen wir nun durch vollständige Induktion, dass

$$n \cdot (2n+1) \cdot \binom{2n}{n} \geq 2^{2n+2}, \forall n \geq 3 \tag{23}$$

gilt.[57]

IA: $n = 3$

$$3 \cdot (2 \cdot 3 + 1) \cdot \binom{6}{3} = 420 > 256 = 2^{2 \cdot 3+2}.$$

IH: Für ein beliebiges $n \geq 3$ sei (23) wahr.

IS:

$$(n+1) \cdot (2n+3) \cdot \binom{2n+2}{n+1} = (n+1) \cdot (2n+3) \cdot \binom{2n}{n} \cdot \frac{2 \cdot (2n+1)}{(n+1)}$$

$$\geq 2 \cdot (n+1) \cdot (2n+3)\binom{2n}{n} \cdot \frac{2n}{(n+1)} = 2 \cdot 2n \cdot (2n+3) \cdot \binom{2n}{n}$$

$$= 4 \cdot n \cdot (2n+3) \cdot \binom{2n}{n} \underbrace{\geq}_{IH} 4 \cdot 2^{2n+2} = 2^2 \cdot 2^{2n+2}$$

$$= 2^{2n+4} = 2^{2 \cdot (n+1)+2}$$

57 Die Aussage ist erst ab $n = 3$ gültig. Für $n = 2, 2 \cdot 5 \cdot \binom{4}{2} = 60 < 64 = 2^6$ und da alle Funktionen monoton wachsend sind gilt die Aussage auch für $n = 1$ nicht.

Somit gilt die Aussage für alle $n \geq 3$.

Daraus folgt $d_{2n+2} \geq d_{2n+1} \geq n \cdot (2n+1) \cdot \binom{2n}{n} \geq 2^{2n+2}$, $\forall n \geq 3$.

Folglich gilt auch $\forall n \in \mathbb{N}$, $n \geq 7$:

$$kgV(n) = d_n \geq 2^n$$

was zu beweisen war. □

Lemma 4.2. *Es existiert ein* $r \leq max\{3, \lceil log^5 n \rceil\}$*, sodass* $o_r(n) > log^2 n$.

Beweis. Der Beweis lässt sich in 3 Schritten zerlegen[58]; beim ersten Schritt handelt es sich darum, zu zeigen, dass ein geeignetes $r \leq B$ existiert, danach wird die Existenz von $o_r(n)$ gezeigt. Als letztes wird die Eigenschaft $o_r(n) \geq log^2 n$ für dieses r bewiesen.

Zunächst sei $n > 1$, für $n = 2$ und $r = 3$ gilt trivialerweise :

$o_3(2) = 2^2 = 4 = 1 (\text{mod } 3) > 1 = log^2 2$.

Nun wird angenommen, dass $n > 2$.[59]

Sei $B = \lceil log^5 n \rceil$, nach Lemma 4.1 gilt $kgV(B) \geq 2^B$. Das Ziel ist zu zeigen, dass ein $r \leq B$ existiert, sodass

$$\Pi = n^{\lfloor logB \rfloor} \cdot \prod_{i=1}^{\lfloor log^2 n \rfloor} (n^i - 1)$$

durch r nicht teilbar ist. Dies kann durch Widerspruch bewiesen werden.

Angenommen, $\forall r$, $1 \leq r \leq B$, r teilt Π, dann gilt $\Pi \geq kgV(B)$. Da Π in diesem Fall ein Vielfaches aller Zahlen kleiner gleich B ist.

Wir betrachten:

$$\begin{aligned}
\Pi = n^{\lfloor logB \rfloor} \cdot \prod_{i=1}^{\lfloor log^2 n \rfloor} (n^i - 1) &< n^{\lfloor logB \rfloor} \cdot \prod_{i=1}^{\lfloor log^2 n \rfloor} n^i \\
&= n^{\lfloor logB \rfloor + \sum_{i=1}^{\lfloor log^2 n \rfloor} i} \\
&= n^{\lfloor logB \rfloor + \frac{log^2 n \cdot (log^2 n + 1)}{2}} 60 \\
&= n^{\lfloor logB \rfloor + \frac{log^4 n + log^2 n}{2}} \qquad (24) \\
&\leq n^{\lfloor logB \rfloor + \frac{log^4 n + \frac{log^4 n}{2}}{2}} \\
&\leq n^{\lfloor \frac{log^4 n}{4} \rfloor + \frac{log^4 n + \frac{log^4 n}{2}}{2}} \\
&\leq n^{log^4 n} = 2^{log\, n^{log^4 n}} = 2^{log^5 n} = 2^B.
\end{aligned}$$

58Der Beweis ist basiert auf den Beweis in [Sta].

59. Der Logarithmus ist eine monoton wachsende Funktion und daher gilt:
$\lceil log^5 3 \rceil = 11 \Rightarrow log^5 n > 10$, $\forall n > 2$.

Aus (24) folgt: $kgV(B) \leq \Pi \leq 2^B \Rightarrow kgV(B) \leq 2^B$. Das ist aber ein Widerspruch zu Lemma 4.1. Das heißt es existiert eine Menge von Zahlen $R = \{r_1, r_2, ..., r_t\}$, $1 \leq r_i \leq B, i = 1, 2, .., t$, sodass alle r_i das Produkt nicht teilen. Sei r das kleinste Element dieser Menge.

Nun wird gezeigt, dass $o_r(n)$ für dieses r existiert[61].

Sei $r = ab$, wobei a aus den Primfaktoren besteht, die n teilen und b aus den restlichen Primfaktoren. Offensichtlich $(b, n) = 1$. Aus der Beobachtung, dass $m^k \leq B$, $m \geq 2$ höchstens $\lfloor logB \rfloor$ ist, folgt, dass $\lfloor logB \rfloor$ die höchste Potenz in der Primfaktorzerlegung von a ist[62]. Da sonst $a \leq r$ größer als B wäre.

$\Rightarrow a \mid n^{logB}$.

$\prod_{i=1}^{\lfloor log^2 n \rfloor}(n^i - 1)$ ist durch b nicht teilbar, da sonst r das Produkt Π teilen würde. b teilt n^{logB} auch nicht, da n und b keine gemeinsamen Primfaktoren haben[63]. r ist das kleinste Element, das Π nicht teilt, deshalb gilt $r = b$. Es gilt außerdem $(b, n) = (r, n) = 1$ und somit existiert $o_r(n)$.

Es bleibt nur noch zu zeigen, dass $o_r(n) > log^2 n$. Das lässt sich auch wie oben durch einen Widerspruch zeigen. Sei $o_r(n) = k \leq log^2 n$. Nach Definition ist k die kleinste Zahl, für die $n^k = 1 \pmod r$ wahr ist. daher gilt auch:

$$n^k - 1 = 0 \,(mod\, r) \Rightarrow r \mid n^k - 1,\ k \leq log^2 n.$$

Aber wenn das gelten würde, würde r das Produkt Π teilen und das führt zum Widerspruch, da $r \nmid \Pi$. Daraus folgt $o_r(n) > log^2 n$ und somit ist das Lemma bewiesen. □

4.2 Laufzeit des AKS-Algorithmus

Zur Berechnung der Zeitkomplexität des AKS-Algorithmus werden die Fakten(2.22) und (2.23) aus dem Zweiten Kapitel verwendet. Das heißt, Multiplikation, und Modulorechnung von zwei Zahlen jeweils mit m Bits können in $O^{\sim}(m)$ Schritten durchgeführt werden. Nach dem gleichen Prinzip können diese Operationen auf zwei Polynome übertragen werden, für Polynome vom Grad d und eine maximale Bitlänge von m können Multiplikationen und Modulorechnungen in $O^{\sim}(d \cdot m)$ Schritten durchgeführt werden.

Theorem 4.3. *Die asymptotische Laufzeit des Algorithmus ist* $O^{\sim}(\|n\|^{10.5})$.

Beweis. Dieses Resultat wird durch Analyse der einzelnen Schritte des Algorithmus festgestellt.

60 Dies folgt aus der gaußschen Summenformel.

61 $o_r(n)$ existiert wenn $(n, r) = 1$

62 Die größte Potenz erhält man, wenn die Basis am kleinsten ist. Das heißt wenn $m = 2 \Rightarrow 2^k \leq B \Rightarrow k \leq logB$.

63 Wegen $(b, n) = 1$

Schritt 1:

Die Laufzeit des ersten Schritts beträgt (laut [GG99] und [Die73]) $O^{\sim}(log^3 n) = O^{\sim}(\|n\|^3)$.

Schritt 2:

Im zweiten Schritt wird ein r gesucht, sodass $o_r(n) > log^2 n$. Dies kann durch Ausprobieren von sukzessiven Werten von r und dann zu testen, ob $n^k \neq 1 \, mod \, r$, $\forall k \leq log^2 n$. Für jedes r gibt es höchstens $O(log^2 n)$ mögliche Werte für k ($O(log^2 n)$ Multiplikationen modulo r). Dies kann in $O^{\sim}(log^2 n \, log \, r)$ berechnet werden.

Gemäß Lemma 4.2 gilt, dass $r \leq log^5 n$, folglich gilt $O^{\sim}(log^2 n \, log \, r) = O^{\sim}(log^2 n \, log \, log^5 n)$, somit gilt, dass die Laufzeit von dem zweiten Schritt $O^{\sim}(log^7 n) = O^{\sim}(\|n\|^7)$ ist[64].

Schritt 3:

Im dritten Schritt wird der größte gemeinsame Teiler von r Zahlen (da $a \leq r$) berechnet.Der ggT von Zahlen der Größe $log \, n$ kann in $O(log \, n)$ berechnet werden[Die73]. Daraus folgt, dass Schritt 3 in $O(r \, log \, n) \underbrace{=}_{4.2} O(log^6 n) = O(\|n\|^6)$ berechnet werden kann[65].

Schritt 4:

Die Zeitkomplexität von $n \leq r$ ist einfach $O(log \, n)$ [GG99].

Schritt 5:

Im fünften Schritt des Algorithmus werden $\ell = \lfloor \sqrt{\phi(n)} log \, n \rfloor$ Gleichungen überprüft. In jeder Gleichung wird $(X+a)^n \, mod \, (X^r - 1, n)$ berechnet und wird danach mit $X^{n \, mod \, r} + a$ verglichen. Gemäß Theorem 2.24 benötigt jede Gleichung $O(log \, n)$ Multiplikationen von Polynomen vom Grad r. Dabei ist die Größe der Polynomkoeffizienten $O(log \, n)$. Somit lässt sich jede Gleichung in $O^{\sim}(r \, log^2 n)$ verifizieren. Daher gilt das folgende für die gesamte Laufzeit der Schleife(ℓ Gleichungen):

$$\begin{aligned} O^{\sim}(\ell \, r \, log^2 n) &= O^{\sim}(r \sqrt{\phi(n)} log^3 n) \\ &= O^{\sim}(r^{3/2} \, log^3 n) \\ &\underbrace{=}_{4.2} O^{\sim}(log^{15/2} n \, log^3 n \\ &= O^{\sim}(log^{21/2} n) = O^{\sim}(\|n\|^{21/2}). \end{aligned}$$

Die Laufzeit des fünften Schritts ist größer als alle verkommenden Laufzeiten. Daher ist die asymptotische Laufzeit des Algorithmus $O^{\sim}(log^{21/2} n) = O^{\sim}(\|n\|^{21/2}) = O^{\sim}(\|n\|^{10.5})$.

64$log \, r$ ist in diesem Fall die Eingabegröße für r.
65Der ggT wird mit dem euklidischen Algorithmus berechent.

□

4.3 Verbesserungen der Laufzeit

In diesem Abschnitt werden mögliche Verbesserungen der Laufzeit des AKS-Algorithmus diskutiert.

Die Laufzeitverbesserung hängt von der Abschätzung für das passende r im zweiten Schritt des Algorithmus ab. Kann man eine kleinere obere Schranke für das Finden des geeigneten r bestimmen, dann lässt sich die gesamte Zeitkomplexität des Algorithmus verringern. Es gibt zwei Vermutungen, die es nahelegen, dass es für das geeignete r tatsächlich eine obere Schranke gibt. Diese Vermutungen sind aber noch nicht bewiesen.

Vermutung 4.4 (**Artins Vermutung**). *Sei $n \in \mathbb{N}$ beliebig aber keine Quadratzahl. Dann gilt, dass die Anzahl der Primzahlen $q \leq m$ mit $o_q(n) = q - 1$ asymptotisch gleich $A(n) \cdot \frac{m}{ln(n)}$ ist. Wobei $A(n) > 0.35$ für die Artin-Konstante steht und ln der natürliche Logarithmus(Basis e) ist.*

Es wurden Fortschritte zum Beweis dieser Vermutung gemacht,beispielsweise in [HEA], oder [Mur]. Sollte die erweiterte Riemannsche Hypothese wahr sein, dann ist Vermutung 4.4 auch richtig.

Die zweite Vermutung beruht auf folgender Definition.

Definition 4.1. Sei p eine Primzahl. p heißt Sophie-Germain Primzahl, wenn $2p+1$ auch eine Primzahl ist. $2p + 1$ heißt in dem Fall eine sichere Primzahl(engl. safe prime).

Vermutung 4.5 (**Häufigkeit von Sophie-Germain-Primzahlen**). *Die Anzahl der Sophie-Germain-Primzahlen bis einer Zahl m lässt sich asymptotisch durch $\frac{2C_2m}{ln^2m}$ abschätzen, wobei C_2 die Zwillingskonstante ist und ungefähr 0.66 beträgt.*

Gemäß Vermutung 4.5 existiert für eine passende Konstante c mindestens log^2n Sophie-Germain Primzahlen zwischen $8log^2n$ und $clog^2n(log\,log\,n)^2$. Für eine Primzahl p und ihre sichere Primzahl $q = 2p + 1$ gilt, dass $o_q(n) \leq 2$ oder $o_q(n) \geq (q - 1)/2$, da $\mathbb{K}_p^*$ die Ordnung $q - 1 = 2p$ hat. Wenn $o_q(n) \leq 2$ gilt, dann gilt $q \mid n^2 - 1$.[66] Daraus folgt, dass die Anzahl der zu testenden q Werte durch $O(log\,n)$ beschränkt ist. Nach Vermutung 4.5 existiert daher eine Primzahl $r = O^{\sim}(log^2n)$, sodass $o_r(n) > log^2n$.

Aus beiden Vermutungen folgt eine bessere Abschätzung für r. Unter der Annahme, dass eine der Vermutungen wahr ist, lässt sich die Zeitkomplexität des AKS-Algorithmus von $O^{\sim}(log^{21/2}n)$ auf $O^{\sim}(log^6n)$ reduzieren. Das geeignete r kann auch ohne Vermutungen begrenzt werden. Fouvry hat in [Fou] ein Lemma vorgestellt, mit dem eine Laufzeit von $O^{\sim}(log^{7.5}n)$ erzielt werden kann.

Sei $P(m)$ der größte Primteiler von m. Dann gilt das folgende Lemma.

[66] $o_q(n) = 1 \Rightarrow q \mid (n-1)$ und wenn $o_q(n) = 2 \Rightarrow q \mid (n^2-1)$. Das heißt q teilt $(n^2-1) = (n-1) \cdot (n+1)$ in beiden Fällen.

Lemma 4.6. *Es existiert ein $c > 0$ und ein n_0, sodass*

$$|\{q \mid q\, ist\, eine\, Primzahl, q \leq x, P(q-1) > q^{\frac{2}{3}}| \geq c \cdot \frac{x}{ln\, x}.$$

$\forall x \geq n_0$.

Nach den Berechnungen in [BH] ist das Lemma für $c \leq 0.6683$ erfüllt. Mit Hilfe von Lemma 4.6 kann die Laufzeit des AKS-Algorithmus auf $O^{\sim}(log^{15/2}n)$ verbessert werden.

Theorem 4.7. *Sei $n \in \mathbb{N}$, $n \geq 2$, dann ist die Laufzeit des AKS-Algorithmus $O^{\sim}(log^{15/2}n)$.*

Beweis. Gemäß Lemma 4.6 kann man für ein q mit $P(q-1) > q^{\frac{2}{3}}$ ein $r = O(log^3 n)$ finden, sodass $o_r(n) > log^2 n$. Daraus ergibt sich die Laufzeit $O^{\sim}(r^{\frac{3}{2}} log^3 n) = O^{\sim}(log^{9/2} n\, log^3 n) = O^{\sim}(log^{15/2} n)$. □

Lenstra und Pomerance haben in [HP] eine modifizierte Version von dem AKS-Algorithmus veröffentlicht, die in $O^{\sim}(log^6 n)$ läuft. Der modifizierte Algorithmus von Lenstra und Pomerance hat eine ähnliche Form wie der AKS-Algorithmus, aber die Ringe in der modifizierten Version werden nicht von Einheitswurzeln erzeugt, sondern von gaußschen Perioden.

Damit lässt sich die Laufzeit des AKS-Algorithmus, ohne auf unbewiesene Vermutungen zurückzugreifen, auf $O^{\sim}(log^6 n)$ verbessern.

Eine weitere mögliche Verbesserung des AKS-Algorithmus wurde von einer der Autoren des AKS-Artikels Agrawal vorgeschlagen, mit deren Hilfe die Laufzeit des AKS-Algorithmus auf $O^{\sim}(log^3 n)$ verbessert werden kann. Seine Vermutung(bekannt als Agrawals Vermutung) beruht auf dem Resultat in [BP].

Vermutung 4.8 (**Agrawals Vermutung**). *Falls r eine Primzahl ist und n nicht teilt, dann gilt*

$$(X-1)^n = X^n - 1\, (mod\, X^r - 1, n) \tag{25}$$

so ist entweder n eine Primzahl oder $n^2 = 1\, (mod\, r)$.

Kayal und Saxena [KS] haben gezeigt, dass Agrawals Vermutung für $r \leq 100$ und $n \leq 10^{10}$ wahr ist.

Wenn Agrawals Vermutung allgemein richtig wäre, dann kann die Laufzeit des AKS-Algorithmus auf $O^{\sim}(log^3 n)$ verbessert werden. Jedoch haben heuristische Resultate von Lenstra und Pomerance darauf hingewiesen, dass Agrawals Vermutung allgemein nicht gilt[AKS02]. Daher bleibt die beste Laufzeit von dem AKS-Algorithmus $O^{\sim}(log^6 n)$. Das heißt aber nicht, dass der AKS-Algorithmus schnell ist bzw. in der Praxis einsetzbar ist.

Der AKS-Algorithmus braucht deutlich länger als viele bekannte probabilistische Primzahltests, um zu entscheiden, ob eine gegebene Zahl eine Primzahl ist oder nicht. Deshalb wird der AKS-Algorithmus nicht in der Praxis benutzt.

Folgende Grafik illustriert, wie langsam der AKS-Algorithmus(dunkelgrün in der Grafik) im Vergleich zu anderen populären Primzahltests läuft[Exc].[67]

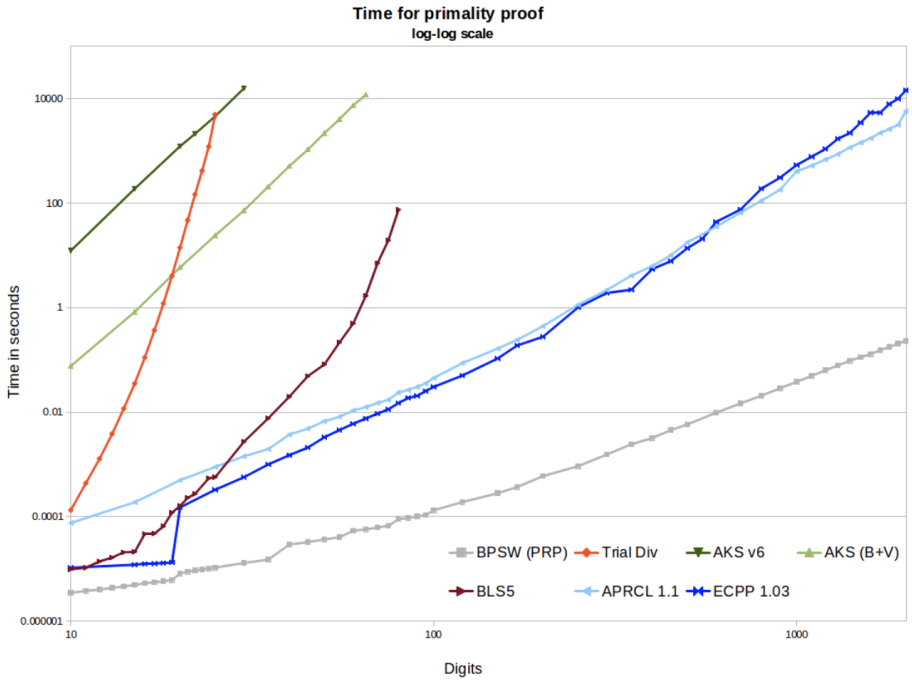

Abbildung 1: Laufzeit des AKS-Algorithmus im Vergleich zu anderen populären Primzahltests

Aus diesem Grund wird der AKS-Algorithmus(oder eine randomisierte Version vom AKS-Algorithmus) in der Praxis nicht benutzt. In der Praxis wird eine schnelle Version vom elliptischen Kurven Primzahltest benutzt, dieser Primzahltest hat eine heuristische Laufzeit von $O^{\sim}(log^4 n)$. er generiert auch ein Zertifikat für die Primalität einer Zahl, das in $O^{\sim}(log^3 n)$ überprüft werden kann[68]. Mit Hilfe dieses Algorithmus wurde Fast jede Primzahl mit mehr als 1000 Stellen, die keine spezielle Form hat (z.B. mersennsche Primzahl), gefunden[Sut17].

67Die Grafik wurde auf dem öffentlichen Forum StackExchange gefunden. Dort gibt es auch eine Erklärung für die Grafik.

68Der elliptische Kurven Primzahltest hat auch mehr konstante Faktoren als der randomisierte AKS[Sut17].

5 Experimentelle Auswertung

In diesem Kapitel werden die Hauptschritte des AKS-Algorithmus(Schritt 2 und 5) untersucht. Zunächst wird überprüft, dass die Laufzeit dieser Schritte polynomiell von der Eingabegröße abhängig ist. Der Fokus liegt hier nicht auf die Korrektheit dieser Schritte(Dies wurde schon im dritten Kapitel gezeigt) sondern auf die polynomielle Laufzeit. Schließlich wird der AKS-Primzahltest mit dem naiven Primzahltest verglichen[69].

5.1 Implementierungsdetails

Zur Implementierung der Algorithmen wurde das Computer-Algebra-System SageMath verwendet. Die Grafiken wurden mithilfe von der öffentlichen Library MatplotLib gemacht. Der Quellcode ist auf meinem Github zu finden. Die Experimente wurden auf einem Computer mit einem Intel Core i7 mit vier Kernen, die jeweils auf 2,8 GHz getaktet sind, und mit 16GB Arbeitsspeicher unter Mac OS Catalina 10.15.3 ausgeführt.[70]

5.2 Experimente

Um die Laufzeit der implementierten Algorithmen auszuwerten, wird zunächst erläutert, wie die Testinstanzen erzeugt werden. Bevor die Erzeugung genauer beschrieben wird, wird auf die verwendete Mengen und die Struktur dieser Mengen eingegangen. Für alle Experimente sind folgende Mengen erforderlich:

1. $I = \{I_1, I_2, \ldots, I_n\}$ Menge der Inputs, die Inputs sind in diesem Fall Arrays von natürlichen Zahlen. I_k beschreibt die Menge der Zahlen, die Bitlänge k haben[71].
2. $T = \{T_1, T_2, \ldots, T_n\}$ Menge der Outputs, hier sind die Outputs Arrays von gemessenen Laufzeiten für die jeweiligen Inputs.
3. $T_{max} = \{max\{T_1\}, max\{T_2\}, \ldots, max\{T_n\}\}$.
4. $I_r = \{k \mid \forall I_k \in I\}$ Menge der eindeutigen Inputgrößen.

Die Testinstanzen bestehen aus den Mengen $I_k \in I, k = 1, 2, \ldots, n$. Wie bereits erwähnt, die Mengen I_k enthält nur natürliche Zahlen mit Bitlänge k. Für alle Experimente ist die Größe dieser Mengen fest und gleich 100. Das heißt $|I_k| = 100, \forall\, k, 1 \leq k \leq n$.

Um eine der Mengen I_k zu generieren, werden zuerst 100 sukzessive Zahlen mit Bitlänge k erzeugt. Diese Zahlen werden danach zufällig aufgeteilt. Die Testinstanzen werden dann durch die Zeitmessung der zu testenden Funktionen auf die Menge T abgebildet. Das bedeutet, es werden immer die Paare $(x, y) \in I \times T$ betrachtet. Für die zwei ersten Experimente wird die folgende Datenmenge verwendet:

$$I = \{I_8, I_9, I_{10}, I_{12}, I_{13}, I_{15}, I_{16}, I_{17}, I_{19}, I_{22}, I_{24}, I_{25}, I_{27}\}.$$

69. Der naive Algorithmus testet ob die eine Zahl n einen Primfaktor $\leq \sqrt{n}$ hat.
70 Alle Zeitmessungen wurden mit der Python-Funktion process_time() gemessen.
71 $I_k = \{n \in \mathbb{N} \mid \|n\| = k\}$.

Notation:

1. Es wird das Symbol Θ für die mittlere Laufzeit(Average-Case-Complexity) verwendet. Im diesem Zusammenhang ist $\Theta(t(n))$ polynomiell, falls $t(n) = log^k n$.
2. Für die Laufzeit(Worst-Case-Complexity) wird wie üblich die groß-O Notation(engl. Big-Oh) verwendet.

5.2.1 Experiment 1: Die Suche nach einem geeigneten r

Einer der Hauptschritte des AKS-Algorithmus ist der zweite Schritt. Im zweiten Schritt wird nach einem r gesucht, sodass $o_r(n) > log^2 n$. Dies passiert durch ausprobieren von mehreren r Werten. In Lemma 4.2 wurde gezeigt, dass dieses r existiert, es wurde außerdem gezeigt, dass $r \leq log^5 n$. In diesem Experiment wird gezeigt, dass die mittlere Laufzeit sowie die Laufzeit(Worst-Case-Complexity) des zweiten Schritts polynomiell von der Eingabegröße abhängig sind.

In diesem Experiment soll die polynomielle Laufzeit des zweiten Schritts vom AKS-Primzahltest demonstriert werden. Hierzu werden verschiedene Inputgrößen betrachtet. Für jede Inputgröße werden 100 zufällig ausgewählte Datenpunkte verwendet, danach wird für jeden Datenpunkt die Laufzeit gemessen. Dadurch erhält man mehrere verstreute Punkte im zweidimensional Raum, an diese Punkte wird ein Polynom angepasst, dieses Polynom soll (Gemäß Kapitel 4) Grad 7 haben. Das Ziel ist eine bestmögliche Approximation(durch ein Polynom) für die Funktion $f(x) = y$ zu berechnen, wobei $x \in I_k$ und $y \in T_k, k = 1, 2, \ldots, n$. Die Approximation der Funktion von Mittelwerten wurde mit der $cruve\_fitting()$ Funktion von scipy durchgeführt. Für die Worst-Case-Laufzeit wurde die Funktion $polyfit()$ benutzt.

Ergebnisse:

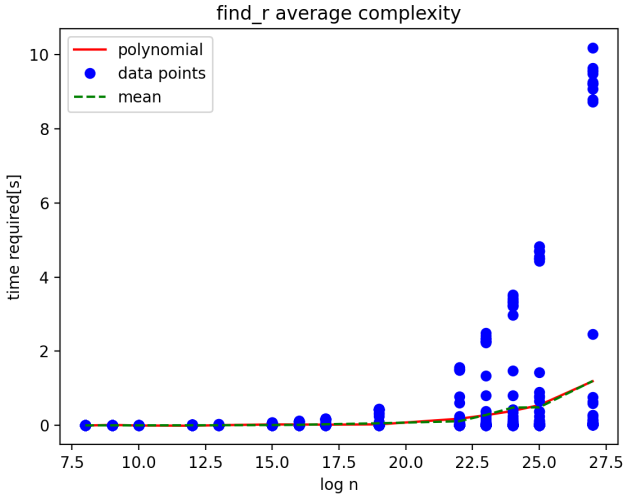

Abbildung 2: Polynomielle mittlere Laufzeit für das Finden eines geeignetes r

Abbildung 2 zeigt, dass das gewählte Polynom vom Grad 7 (rot) die Funktion der Mittelwerte von den Inputgrößen(grün) nahezu zu perfekt darstellt. Mit Hilfe dieser empirischen Evidenz kann man nun folgern, dass die Average-Case-Complexity $\Theta(t(n))$ polynomiell ist[72]. Das heißt $\Theta\tilde{}(log^k n), k > 0$. In diesem Fall ist $k = 7$.

mit Hilfe der Datenpunkte im ersten Experiment kann auf ähnliche Weise auch den schlechtmöglichsten Fall(Worst-case-Complexity) bestimmt werden. Hierzu wird das Polynom an die Maximumpunkte von T_k, $k = 1, 2, \ldots, n$ angepasst. Das bedeutet, folgende Funktion wird nun approximiert:

$$f(x) = y, \; wobei \, y \in T_{max}, x \in I_r.$$

Zu erwarten ist, dass das Polynom die Maximumpunkte nahezu perfekt passiert. Da im vierten Kapitel gezeigt wurde, dass die Laufzeit des zweiten Schritts polynomiell ist. Genauer: die Laufzeit ist durch ein Polynom siebten Grads von oben beschränkt.

Ergebnisse:

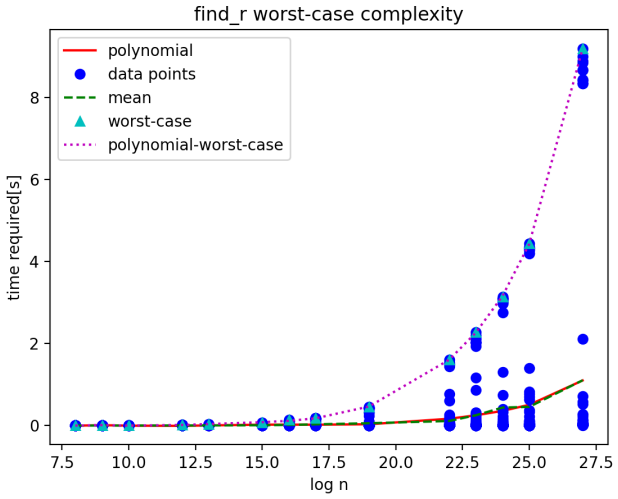

Abbildung 3: Polynomielle Laufzeit für das Finden eines geeignetes r(Worst-Case)

Es ist leicht zu sehen, dass das Polynom (mit Grad 7) durch die Maximumpunkte bei jeder Eingabe durchströmt. Dies ist gerade eine empirische Evidenz für die polynomielle Laufzeit des zweiten Schritts des AKS-Algorithmus.

72Da die mittlere Laufzeiten durch ein Polynom von $log\, n$ beschränkt sind.

5.2.2 Experiment 2: Laufzeit des AKS-Algorithmus

In diesem Experiment wird die polynomielle Laufzeit des AKS-Algorithmus überprüft. Im vierten Kapitel wurde gezeigt, dass der AKS-Algorithmus eine polynomielle Laufzeit hat ($O^{\sim}(log^{10.5}n)$), dies wird in diesem Experiment getestet. Zuerst wird die mittlere Laufzeit geprüft, und danach wird dies allgemein für die Laufzeit (Worst-Case-Complexity) geprüft.

In diesem Experiment wird auch wie beim ersten Experiment ein Polynom(diesmal aber vom Grad 10.5) an die Datenpunkte angepasst, die Datenpunkte sind wieder Paare $(x, y) \in I \times T$. Hier wird auch für jede Eingabegröße 100 Datenpunkte ausgewählt.

Ergebnisse:

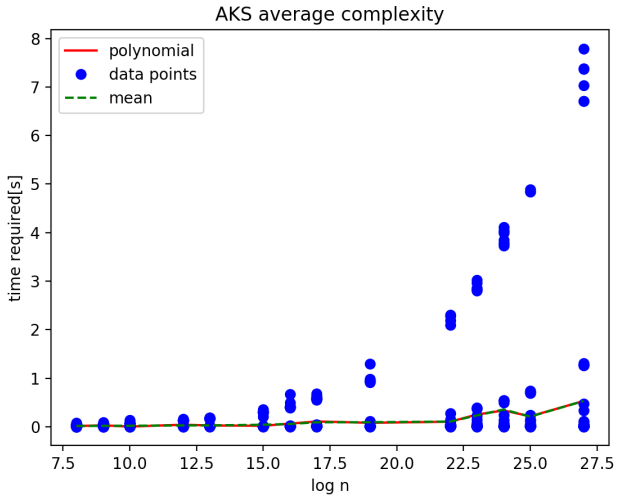

Abbildung 4: Mittlere polynomielle Laufzeit des AKS-Algorithmus

Abbildung 4 zeigt deutlich, dass das Polynom durch alle Mittelwerte (für alle Eingabegrößen) durchgeht. Daraus folgt, dass die mittlere Laufzeit polynomiell von der Eingabegröße abhängig ist.

Mit den selben Datenpunkten (x, y) lässt sich auch zeigen, dass die Worst-Case-Laufzeit polynomiell ist. Dies kann durch die Anpassung eines Polynoms mit Grad 10.5(Siehe Kapitel 4) an die höchsten Laufzeiten bei jeder Eingabegröße, erreicht werden.

Das erwarte Verhalten hier ist, dass das Polynom durch alle Maximumpunkte durchströmen soll, dies wäre ein empirischer Beweis dafür, dass die Laufzeit des AKS-Algorithmus polynomiell ist(abhängig von der Eingabegröße). Genauer: die Laufzeit kann durch ein Polynom von $log\, n$ mit Grad 10.5 beschrieben werden.

Ergebnisse:

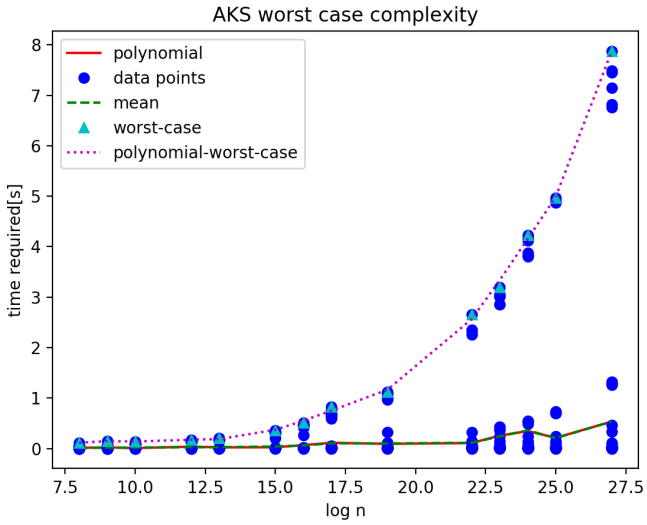

Abbildung 5: Polynomielle Laufzeit des AKS-Algorithmus

In Abbildung 5 ist leicht zu sehen, dass das angepasste Polynom(Rosa) durch alle Maximumpunkte durchgeht.

Abbildung 5 Liefert also einen empirischen Beweis dafür, dass die Laufzeit des AKS-Algorithmus durch ein Polynom mit Grad 10.5 begrenzt ist. Mit anderen Worten, dass die Laufzeit des AKS-Algorithmus polynomiell ist.

5.2.3 Experiment 3: AKS vs naiver Primzahltest

In der Einführung dieser Arbeit wurde ein Algorithmus(naiver Primzahltest) erwähnt, der das Primalitätsproblem in $\Omega(\sqrt{n})$ löst[73]. Allerdings hat dieser Primzahltest eine exponentielle Laufzeit. Der AKS-Primzahltest dahingegen hat eine polynomielle Laufzeit, und ist somit schneller als der naive Primzahltest. Dies wird in diesem Experiment demonstriert, hierzu wurden die Laufzeiten beider Algorithmen bis zur Inputgröße 37 verglichen.

Zur Untersuchung der Laufzeit beider Algorithmen als Primzahltest werden bewusst nur Primzahlen als Eingabewerte herangezogen. Das liegt daran, dass diese den schlechtest möglichen Fall beider Algorithmen darstellt.

Wie bereits erwähnt hat der naive Primzahltest eine exponentielle Laufzeit, das heißt aber nicht, dass der naive Algorithmus für alle Inputgrößen langsamer als der AKS-Primzahltest ist. Es kann sein, dass der naive Algorithmus z.B. für kleine Inputgrößen eine schnellere Antwort als der AKS-Algorithmus liefert.

Folgende Tabelle ist einerseits ein sehr schönes Beispiel für die immense Steigerung der Laufzeit des naiven Primzahltests. Andererseits ist sie auch ein Beispiel für den Fall, wo der naive Primzahltest schneller als der AKS-Primzahltest ist.

AKS vs Naive			
Zahl	Inputgröße	AKS [in s]	Naive [in s]
8191	13	0.319566	0.0006
131071	17	0.990821	0.01
524287	19	1.5811	0.04
38757413	26	7.556	2.933
2147483647	31	42.58	162.3
2547587681	32	43.902	187.876
17014120163	34	86.073	1275.286
90552556889	37	97.769072	6732.12

Tabelle 1: AKS vs naiver Primzahltest.

Obige Tabelle macht auch deutlich, dass der AKS-Primzahltest eine bessere asymptotische Laufzeit als der naive Primzahltest hat. Es ist aber augenfällig, dass der AKS-Primzahltest für Inputgrößen kleiner gleich 26 langsamer als der naive Primzahltest ist. Das liegt daran, dass die Laufzeit des AKS-Primzahltest $O^{\sim}(log^{10.5}n)$ beziehungsweise $O^{\sim}(log^{7}n)$(wenn $n \leq r$) für "kleine" Inputgrößen größer oder gleich als die Laufzeit des naiven Primzahltests $O(\sqrt{n})$ ist. Dies lässt sich am besten durch die folgende Grafik illustrieren:

73Für eine Erklärung, warum ein Algorithmus mit $\Omega(\sqrt{n})$ im Kontext von Primalitätstesten eine exponentielle Laufzeit hat siehe: `https://cs.stackexchange.com/questions/88398/primality-testing-why-is-dividing-a-number-n-by-every-integer-between-2-and`.

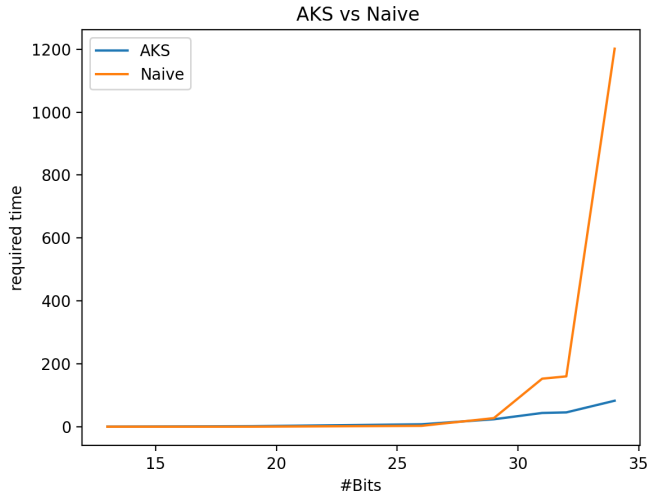

Abbildung 6: Laufzeit des AKS-Algorithmus im Vergleich zum naiven Algorithmus

In der Grafik 6 sieht man, dass der AKS-Primzahltest bei Inputgröße 37 in unter $200\,s$ eine Antwort geliefert hat, während der naive Primzahltest mehr als $6500\,s (> 1.5h)$ brauchte.[74] Aber bei Inputgrößen bis ungefähr 29 ist die Laufzeit beider Algorithmen fast identisch.

74In der Tabelle wurde die Laufzeit von Zahlen bis Inputgröße 37 gemessen, während in der Grafik nur Zahlen mit Inputgröße bis 35 zu sehen sind. Das liegt einerseits daran, dass die Grafik für diese Werte übersichtlicher ist. Andererseits sind diese ausreichend, um zu zeigen, dass die asymptotische Laufzeit des AKS-Algorithmus wesentlich besser ist.

6 Ausblick und Zusammenfassung

Zur Anfang der Arbeit wurde der Stand vom Primalitätstesten vor dem AKS-Algorithmus kurz dargestellt. Danach wurden der AKS-Algorithmus und seine Grundidee im Detail beschrieben. Hierzu gehörte unter anderem die Identität, auf der der AKS-Primzahltest beruht. Diese Identität war der Ausgangspunkt dieser Arbeit. Zuerst wurden bestimmte Anforderungen für die Variablen a, r Vorausgesetzt, die unbedingt für die Polynomiallaufzeit des AKS-Algorithmus erforderlich sind. Mithilfe von diesen Voraussetzungen wurde eine obere Schranke für die Anzahl der zu testenden a Werte hergeleitet.

Nach der Vorstellung des AKS-Algorithmus und seine nötigen Voraussetzungen wurde der Korrektheitsbeweis durchgeführt. Im Beweis wurde gezeigt, dass der Algorithmus als ein Primzahltest verwendet werden kann, das heißt der Algorithmus liefert immer 1 (PRIME), wenn die Eingabe n eine Primzahl ist, und wenn die Ausgabe 1 (PRIME) ist, dann ist die eingegebene Zahl n tatsächlich eine Primzahl.

Nachdem die Korrektheit des AKS-Algorithmus bewiesen wurde, wurde die Laufzeitanalyse dargestellt. Zunächst wurde eine Abschätzung für das geeignete r gemacht, die für die Laufzeitanalyse von Bedeutung ist. Die Laufzeitanalyse ergab, dass der AKS-Algorithmus eine polynomielle Laufzeit hat. Es wurden außerdem mehrere Verbesserungen für die Laufzeit vorgeschlagen, einige Verbesserungen waren auf Vermutungen basiert. Es wurde aber auch einen anderen Ansatz erwähnt, mit dem sich die Laufzeit $O^{\sim}(log^{10.5}n)$, ohne auf Vermutungen zurückgreifen zu müssen auf $O^{\sim}(log^{6}n)$ reduzieren lässt. Im gleichen Kapitel wurde auch kurz diskutiert, warum der AKS-Algorithmus nicht in der Praxis einsetzbar ist.

Es wurde mit Hilfe von mehreren Experimenten demonstriert, dass die Hauptschritte (Schritt 2 & 5) des AKS-Algorithmus polynomiell von der Eingabegröße abhängig sind, und dass somit die Laufzeit des AKS-Algorithmus auch polynomiell ist. Schließlich wurde der AKS-Algorithmus mit dem naiven Algorithmus (Probedivision Algorithmus) verglichen und es hat sich herausgestellt, dass der AKS-Algorithmus eine bessere asymptotische Laufzeit hat[75].

Zusammenfassend: Der AKS-Primzahltest ist ein wichtiger theoretischer Durchbruch im Bereich des Primalitätstestens, er war der erste deterministische bedingungslose Algorithmus mit polynomieller Laufzeit, der das Primalitätsproblem löst. Aber in der Tat ist der AKS-Algorithmus Trotz seiner erstaunlichen Eigenschaften immer noch für große Primzahlen sehr langsam. Daher wird weder der AKS-Primzahltest noch seine randomisierte Version in der Praxis benutzt.

75Das ist auch natürlich richtig, da die Laufzeit des naiven Algorithmus exponentiell ist, während die Laufzeit des AKS-Algorithmus polynomiell ist.

Literatur

[Die73] Martin Dietzfelbinger. *Primality Testing in Polynomial Time*. 1973.

[Mil75] G. L. Miller. *Riemann's hypothesis and tests for primality, J. Comput. Sys. Sci. 13 (1976), 300–317*. 1975. URL: https://www.sciencedirect.com/science/article/pii/S0022000076800438.

[SS77] R. Solovay und V. Strassen. *A Fast Monte-Carlo Test for Primality*. 1977. URL: https://epubs.siam.org/doi/abs/10.1137/0206006?mobileUi=0&.

[Nai82] M. Nair. *On Chebyshev-Type Inequalities for Primes*. 1982. URL: https://www.jstor.org/stable/2320934?seq=1.

[Lid86] Rudolf Lidl. *Introduction to finite fields and their applications*. 1986.

[Pap93] Christos Papadimitriou. *Computational Complexity*. 1993.

[GG99] Joachim von zur Gathen und Jürgen Gerhard. *Modern Computer Algebra, second Edition*. 1999.

[AKS02] Manindra Agrawal, Neeraj Kayal und Nitin Saxena. *PRIMES is in P*. 2002. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/primality_journal.pdf.

[CP05] Richard Crandall und Carl Pomerance. *Prime Numbers, A Computational Perspective, Second Edition*. 2005.

[Oli09] Sebastian Schonnenbeck Oliver Braun. *Der AKS-Primzahltest, Vortrag im Rahmen des Proseminars zur linearen Algebra*. 2009. URL: http://www.math.rwth-aachen.de/~Gabriele.Nebe/Vorl/pros/AKS.pdf.

[Aka12] Fred Akalin. *Notes on math, tech, and everything in between*. 2012. URL: https://www.akalin.com/.

[Sut17] Andrew Sutherland. *Lecture Series in Mathematics*. 2017. URL: https://math.mit.edu/~drew/PrimesInP.pdf.

[AB] Sanjeev Arora und Boaz Barak. *Computational Complexity: A Modern Approach*. URL: https://theory.cs.princeton.edu/complexity/book.pdf.

[BH] R. C. Baker und G. Harman. „The Brun-Titchmarsh Theorem on average, in Analytic Number Theory, Volume I (Allerton Park, IL, 1995), Progr. Math. 138, 39–103, Birkhauser Boston, Boston, MA, 1996“. In: ().

[bar] faculty bard. *Fields and Cyclotomic Polynomials*. URL: http://faculty.bard.edu/~belk/math318/CyclotomicPolynomials.pdf.

[BP] R. Bhattacharjee und P. Pandey. „, Primality testing, Technical report, IIT Kanpur, 2001; available at http://www.cse.iitk.ac.in/research/btp2001/primality.html.“ In: ().

[Exc] Stack Exchange. *When is the AKS primality test actually faster than other tests?* URL: https://cs.stackexchange.com/questions/23260/when-is-the-aks-primality-test-actually-faster-than-other-tests.

[Fou] E. Fouvry. „Theor'eme de Brun-Titchmarsh; application au theor'eme de Fermat, Invent. Math. 79 (1985), 383–407". In: ().

[HP] Jr. H. W. Lenstra und C. Pomerance. „Primality testing with gaussian periods". In: ().

[HEA] HEATH-BROWN. *ARTIN'S CONJECTURE FOR PRIMITIVE ROOTS.* URL: https://academic.oup.com/qjmath/article-abstract/37/1/27/1515517.

[KS] N. Kayal und N. Saxena. „Towards a deterministic polynomial-time test, Technical report, IIT Kanpur, 2002; available at http://www.cse.iitk.ac.in/ research/btp2002/primality.html." In: ().

[Kur] Piyush P Kurur. *Primality is in NP coNP.* URL: https://www.cmi.ac.in/~ramprasad/lecturenotes/comp_numb_theory/lecture17.pdf.

[Mur] Rajiv Gupta M. Ram Murty. *A remark on Artin's conjecture.* URL: https://link.springer.com/article/10.1007%2FBF01388719.

[Pin] Charles C. Pinter. *A Book of Abstract Algebra.* URL: http://www2.math.umd.edu/~jcohen/402/Pinter%20Algebra.pdf.

[Sta] StackExchange. *On proof of AKS primality test algorithm.* URL: https://math.stackexchange.com/questions/119573/on-proof-of-aks-primality-test-algorithm.

[Wika] Wikipedia. *Euler's theorem.* URL: https://en.wikipedia.org/wiki/Euler%27s_theorem.

[Wikb] Wikipedia. *Fundamental$_t$heorem$_o$f$_a$rithmetic.* URL: https://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic.

[Wikc] Wikipedia. *Kryptographie.* URL: https://de.wikipedia.org/wiki/Kryptographie.

[Wikd] Wikipedia. *Largest known prime number.* URL: https://en.wikipedia.org/wiki/Largest_known_prime_number.

[Wike] Wikipedia. *Miller–Rabin primality test.* URL: https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test.

[Wikf] Wikipedia. *Monoid.* URL: https://en.wikipedia.org/wiki/Monoid.

[Wikg] Wikipedia. *Polylogarithmic function.* URL: https://en.wikipedia.org/wiki/Polylogarithmic_function.

[Wikh] Wikipedia. *Prime number.* URL: https://en.wikipedia.org/wiki/Prime_number.

[Wiki] Wikipedia. *RSA (cryptosystem)*. URL: https://en.wikipedia.org/wiki/RSA_(cryptosystem).

[Wikj] Wikipedia. *S3 (Gruppe)*. URL: https://de.wikipedia.org/wiki/S3_(Gruppe).

[Wikk] Wikipedia. *Schriftliche Division*. URL: https://de.wikipedia.org/wiki/Schriftliche_Division.

A Algorithmenverzeichnis

Algorithm 2 Potenz-Prüfung

Input: $n \in \mathbb{N}, n \geq 2$.

1. a, b, c, m
2. $b = 2$.
3. **while** $2^b \leq n$ **repeat**
4. $\quad a = 1$, $c = n$
5. $\quad$ **while** $c - a \geq 2$ **repeat**
6. $\quad\quad m = \lfloor (a + c)/2 \rfloor$
7. $\quad\quad p = min\{m^b, n + 1\}$
8. $\quad\quad$ **if** $p = n$ **return** True.
9. $\quad\quad$ **if** $p < n$
10. $\quad\quad\quad a = m$
11. $\quad\quad$ **else**
12. $\quad\quad\quad c = m$
13. $\quad b = b + 1$
14. **return** False

Algorithm 3 Schnelle modulare Exponentiation

Input: $a \in M, n \geq 0$

1. $u = n$.
2. $s = a$.
3. $c = 1$.
4. **while** $u \geq 1$ **repeat**
5. $\quad$ **if** $u \, mod \, 2 \neq 0$ **then** $c = c \circ s$
6. $\quad s = s \cdot \, mod \, M$
7. $\quad u = u \, div \, 2$
8. **return** c

Abbildungsverzeichnis

Tabellenverzeichnis

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

Leipzig, den 7. Juni 2020 Salman Salman