Berühmte Sätze und Probleme Rund um Primzahlen

Ilja Kononenko und David Silvan Zingrebe

18. Juni 2018

Seminar: Diskrete Mathematik Bei: Heinz-Juergen Schmidt

Inhaltsverzeichnis

Vorwort

Primzahlen sind ein wichtiger Bestandteil der Mathematik, wie wir sie heute kennen. Allerdings werden Primzahlen von den meißten Mathematikern als zufällig und unwichtig abgetan. Allein schon die Tatsache, daß man aus Primzahlen alle nicht primen Zahlen zusammensetzen kann, ist berauschend. Dieser Fakt wird im Kapitel über die Primfaktorzerlegung dieser Seminar genauer beschrieben. Ebenfalls taucht die Frage auf, ob es denn unendlich viele Primzahlen gibt. Dies wird im Kapitel über den Satz des Euklid abgearbeitet. Ist eine Zahl n prim oder nicht? Das ist mit lediglich Stift und Papier nur sehr schwer herauszufinden und mit immer größer werdenden Zahlen stellt sich der Prozess als immer umständlicher und mühsamer heraus. Deshalb behandeln wir in dieser Seminararbeit auch zahlreiche Algorithmen, um herauszufinden, ob eine Zahl prim ist. Ebenfalls behandeln wir Algorithmen zum Finden ganzer Primzahlmengen in einem angegebenen Intervall. Doch Primzahlen sind nicht einfach nur Primzahlen. Es gibt verschiedene Arten von Primzahlen, die unterschiedlich zusammengesetzt sind. Mersenne Primzahlen sowie Fermatsche Primzahlen werden in dieser Seminararbeit genauer beschrieben. Wie in allen anderen Fachgebieten gibt es auch zu Primzahlen vielerlei Behauptungen, die bis heute noch nicht bewiesen worden sind. Dazu gehören die Primzahlzwillinge sowie die Goldbachsche Vermutung. In einigen Kapiteln sind Programmbeispiele zur Veranschaulichung der Themen beigelegt und beschrieben.

1 Definition und Grundlagen von Primzahlen

Primzahlen sind Zahlen welche genau zwei Teiler haben. Sie sind deshalb ein sehr interessanter und wichtiger Bestandteil der Mathematik, obwohl sie nur im natürlichen Bereich der Zahlen eine große rolle Spielen.

1.1 Primfaktorzerlegung

1.1.1 Definition

Jede natürliche Zahl n kann als Produkt von Primzahlen p, den sogenannten Primfaktoren, dargestellt werden. Diesen Vorgang nennt man Primfaktorzerlegung. Die Reihenfolge der einzelnen Primfaktoren p spielt wie bei jeder anderen Multiplikation keine Rolle. Für den Fall, daß n eine Primzahl ist, ist sie selbst ihr einziger Faktor. Bei der Primfaktorzerlegung können Faktoren mehrfach auftreten. Diese kann man Exponentiell zusammenfassen. Kanonische Primfaktorzerlegung wird eine Primfaktorzerlegung genannt, sobald die einzelnen Faktoren nach der Höhe ihrer Basis aufsteigend geordnet sind ($p_k < p_{k+1}$).

1.1.2 Beispiele

$14 = 2 \cdot 7$
$69 = 3 \cdot 23$
$666 = 2 \cdot 3 \cdot 3 \cdot 27$
$1337 = 7 \cdot 191$

Und in der Kanonischen Primfaktorzerlegung:

$666 = 2 \cdot 3^2 \cdot 27$

1.1.3 Faktorisierungsverfahren

Bis heute gibt es kein effizientes Faktorisierungsverfahren. Die einfachste Möglichkeit ist es, die zu faktorisierende, natürliche Zahl n durch alle Primzahlen von Zwei bis zur Wurzel von n zu teilen, bis man einen Teiler gefunden hat, bei dem das Ergebnis keinen Rest hat. Man merkt sich nun diesen Teiler, also den ersten Primfaktor, und führt nun fort, ersetzt jedoch n mit dem gerade errechneten Quotienten. Fortgefürht wird dies, solange n keine Primzahl ist. Ist sie es, hat man seine natürliche Zahl n erfolgreich ausschließlich mit Primzahlen faktorisiert.

1.1.4 Programmbeispiel

Das folgende Programm in C++ generiert eine Abbildung als String der Primfaktorzerlegung der Zahlen von 2 bis g_maxVal.

```
#include <vector>
#include <iostream>
#include <sstream>
#include <thread>
#include <map>
#include <fstream>

const size_t g_maxVal = 0xFFFFFF;

size_t teile(size_t zahl)
```

```
{
        size_t teiler = 2;
        size_t x = sqrt(zahl);
        while (zahl % teiler != 0)
        {
                if (teiler >= x)
                {
                        return zahl;
                }
                else
                {
                        ++teiler;
                }
        }

        return teiler;
}

std::string primfaktorzerlegung(size_t zahl)
{
        std::stringstream ss;
        size_t teiler = 0;

        ss << zahl << " = ";

        while (zahl != 1)
        {
                teiler = teile(zahl);
                ss << teiler;
                zahl = zahl / teiler;
                if (zahl != 1)
                        ss << "*";
        }

        return ss.str();
}

void generate_prime_factor(
        unsigned threadnum,
        unsigned threadc,
        std::map<size_t, std::string>* factors)
{
        if (threadnum == 0)
                threadnum += threadc;
        for (size_t i = threadnum; i < g_maxVal && !g_shouldExit;
                i += threadc)
        {
                factors->insert(
                        std::pair<size_t, std::string>(
                                i, primfaktorzerlegung(i)));
```

```
        }
}

void multithreaded_factorization(
        std::map<size_t, std::string>* factors)
{
        const size_t threadc = std::thread::hardware_concurrency();
        std::thread* threads = new std::thread[threadc];
        for (unsigned i = 0; i < threadc; ++i)
        {
                threads[i] = std::thread(
                        generate_prime_factors,
                        i, threadc, factors);
        }

        for (unsigned i = 0; i < threadc; ++i)
        {
                threads[i].join();
        }

        delete[] threads;
}

void print_factors(     std::ostream& os,
                        std::map<size_t, std::string>* factors)
{
        for (std::map<size_t, std::string>::iterator it =
                        factors->begin();
                        it != factors->end(); ++it)
        {
                os << it->second << "\n";
        }
}

int main(  )
{
        std::map<size_t, std::string> factors;
        multithreaded_factorization(&factors);
        std::ofstream os("testfile.txt",
                std::ios::trunc | std::ios::binary );
        if (os)
                print_factors(os, &factors);
        os.close();
        return 0;
}
```

In der main-Methode wird ein Array erstellt, in dem die Ergebnisse der Primfaktorzerlegung als String abgespeichert werden kann. Anschließend wird die Methode **multithreaded_factorization** aufgerufen, welche, um die Performance zu verbessern, mehrere Threads, deren Anzahl der Anzahl der Kerne der CPU entspricht, gestartet. Diese sollen die jeweiligen Faktorisierungen aller natürlichen Zahlen von 1 bis **g_maxVal** berechnen und in **factors** abspeichern.
Die errechneten Faktoren können mittels der **print_factors**-Methode in einen beliebigen stringstream **os** gestreamt werden. Dies können Dateien, Netzwerkübertragungen oder lediglich das Terminal des Computers sein.
Die Funktion **generate_prime_factors** durchläuft eine Schleife, welche die Primfaktorzerlegung der natürlichen Zahl **threadnum** berechnet und diesen Wert daraufhin um die Anzahl aller Threads **threadc** erhöht. Dadurch kommen sich die einzelnen Threads beim Abspeichern der errechneten Faktoren nicht in die Quere.
Die **primfaktorzerlegun**-Funktion gibt eine Stringrepräsentation der Faktorisierung der übergebenen natürlichen Zahl **zahl** zurück. Dabei wird solange **zahl** nicht eins, also der ersten Zahl, bei der eine weitere Faktorisierung keinen Sinn mehr macht, ist, der kleinste Teiler von **zahl** bestimmt, der eine Primzahl ist und an den zureckzugebenen String angehängt.
Der kleinstmögliche Teiler, der eine Primzahl ist, kann mit der Funktion **teile** bestimmt werden. Der gesuchte Teiler wird ermittelt, in dem geschaut wird, ob die übergebene Zahl dividiert mit dem Startwert 2 des Teilers **teiler** eine natürliche Zahl ergibt. Falls ja, wurde der kleinste Faktor gefunden. Falls nicht wird **teiler** inkrementiert und es wird erneut auf eine Division ohne Rest getestet. Ist **teiler** größer als die Quadratwurzel der gesuchten Zahl **zahl**, kann abgebrochen werden, da **zahl** prim ist und somit ihren bestmöglichen eigenen Faktor darstellt.

1.2 Der Satz des Euklid

1.2.1 Definition

Der Satz des Euklid besagt, daß es unendlich viele Primzahlen gibt.

1.2.2 Geschichte

Wie man dem Namen entnehmen kann, wurde dieser Satz vom griechischem Mathematiker Euklid, der im 3. Jahrhundert nach Christus lebte, aufgestellt.

1.2.3 Beweis des Satz des Euklid

Geht man von einer endlichen Menge Primzahlen p_0 bis p_n aus, deren Produkt v ist, gibt es zwei mögliche Ereignisse für $v + 1$. Das Ergebnis kann prim oder nicht prim sein. Ist $v + 1$ prim, so ist bereits bewiesen, daß die bisher bekannten Primzahlen nicht die einzigen sind und es noch weitere gibt. Ist $v + 1$ hingegen nicht prim, so kann man dessen Primfaktorzerlegung bilden. Diese Primfaktoren p_0 bis p_n enthalten nun einen Faktor q, welcher sowohl von v, als auch von $v+1$ ein Teiler ist. Die einzige Möglichkeit für q ist also 1, was jedoch keine Primzahl ist. Demnach ist die Aussage, daß es endlich viele Primzahlen gibt, widersprüchlich und falsch. Man kann schließen, daß es unendlich viele Primzahlen gibt.

1.2.4 Anwendung

Der Satz des Euklid findet Anwendung bei der Findung weiterer Primzahlen. Kennt man bereits eine oder mehrere Primzahlen, multipliziert man diese miteinander und addiert 1 auf das erhaltene Produkt und es gibt zwei Möglichkeiten die auftreten können. Entweder ist das Ergebnis prim und nicht in der Menge der bereits bekannten Primzahlen enhalten, da sie mindestens 1 größer ist, oder man bildet die Primfaktorzerlegung des Ergebnisses. Mindestens einer dieser Faktoren ist eine bisher noch unbekannt Primzahl.

1.2.5 Beispiele

Man besitze eine endliche Menge an Primzahlen $M_p\{3; 5; 11\}$, multipliziert diese miteinander und addiert eins.
$3 \cdot 5 \cdot 11 + 1 = 166$
Von dem Ergebnis bildet man nun die Primfaktorzerlegung und erhält neue Primzahlen.
$166 = 2 \cdot 83$
Die neuen Primzahlen sind 2 sowie 83.
Besitzt man beispielsweise jedoch nur die Primzahl 3, so addiert man lediglich 1 und wendet wieder die Primfaktorzerlegung an. $3 + 1 = 4 \Rightarrow 4 = 2^2$. Somit erhält man die zuvor noch unbekannt Primzahl 2.

1.3 Simpler Primrechner

Mit den Informationen, die wir bis jetzt erhalten haben, kann man bereits ein simples Programm schreiben, welches alle Primzahlen bis zu einer gegebenen natürlichen Zahl **Maxwert** findet.

```
Vector<Integer> g = new Vector<Integer>();
for (int i = 3; i< Maxwert; i++){
boolean isPrime = true;
                        for (int j = 2; j < Math.sqrt(i); j++) {
                                if (i%j==0) {
                                        isPrime = false;
                                        break;
                                }
                        }
                if(isPrime){
                        g.add(i);
                        }
                }
```

Am Anfang muss man einen Behälter variabler Größe festlegen, da die Anzahl der Primzahlen bis **Maxwert** zu Beginn noch nicht bekannt ist. Anschließend wird eine for-Schleife durchlaufen, welche die zu testende Zahl gibt. Hier wird dann eine Boolsche Variable gesetzt, mit Hilfe welcher später abgefragt wird, ob dieser Test erfolgreich war oder nicht. Die Zahl i wird durch eine for-Schleife mit allen Zahlen von 2 bis zu ihrer Wurzel geteilt. Hat diese Division mit einer Zahl keinen Rest so ist klar, daß i nicht prim sein kann, die Boolsche Variable wird auf false gesetzt und aus der for-Schleife rausgebreakt. Hiermit wird verhindert, daß das Programm unnötig weiterprüft. Man muss dies nur bis zur Wurzel durchführen, da das Quadrieren der Wurzel der Zahl selbst gibt und somit wäre sie durch sich selbst teilbar und höher muss man nicht gehen da es laut der Primfaktorzerlegung einen kleineren Teiler bereits gegeben haben muss. Schlussendlich wird die Boolsche Variable abgefragt und wenn sie True ist wird die Primzahl in den Behälter geschrieben. Dieses Testverfahren wird mit jeder Zahl, bis zu einem vorbestimmten Maximum, durchgeführt. Der Benutzer könnte danach alle Primzahlen simple mit einer for-Schleife abfragen und sie wo immer er will ausgeben oder verwenden. Das Problem mit diesem Verfahren ist das es alle Zahlen durchgeht bis zu dem Maximalwert dadurch läuft es recht langsam.

1.4 Das Sieb des Eratosthenes

Jetzt komme ich auf das Sieb des Eratosthenes zu sprechen, welches ungefähr im Jahre 200 vor Christus vom namensgebenden Mathematiker Eratosthenes entwickelt wurde. Es ist genau wie der simple Primrechner ein Verfahren, um eine große Menge nach Primzahlen zu untersuchen und diese herauszufiltern. Jedoch hat das Sieb des Eratosthenes einen großen Vorteil: seine Effizienz. Dazu kann jeder dieses Prinzip verwenden, wenn er genug Speicherkapazität hat, wobei mit dieser nicht nur die elektrische Form gemeint ist, sondern jede Art von Speicherung. Zum Beispiel könnte ein Blatt Papier als Speicher fungieren. Es funktioniert, indem die Zahl '2' genommen wird und diese automatisch Prim ist. Daraufhin werden alle Multiplikationen, beginnend mit dem Quadrat, dieser Zahl gestrichen. Die nächste Zahl 'n' welche nicht gestrichen ist, wird dann automatisch Prim und der Vorgang des Streichens wird mit dieser wiederholt. Dieser weitere Vorgang wird solange gemacht, bis man zu der Stelle gelangt wo das Quadrat der aktuellen Zahl 'n' über dem Maximum liegt und dann werden alle noch nicht gestrichenen Zahlen in der Liste automatisch Primzahlen.

1.4.1 Programm

Ich zeige Ihnen dieses System jetzt als Veranschaulichung anhand des folgenden Java Programms:

```
private static boolean[] eratosthenes(int max)
        {
                boolean[] primes = new boolean[max];
                for (int i = 2; i < max; ++i)
                {
                        primes[i] = true;
                }

                int i = 2;
                for(; i * i < max; ++i)
                {
                        if (primes[i])
                        {
                                //System.out.println(i);
                                for (int j = i * i; j < max; j = j + i)
                                        primes[j] = false;
                        }
                }


                return primes;
        }
```

1.4.2 Programmerklärung

Das Programm erstellt ein Array aus Boolwerten, welches so groß ist wie ein vorreingetragener Maximalwert, wobei die Indizes für die jeweiligen Zahlen stehen. Da 1 und 0 keine Primzahlen sind werden beginnend mit dem Index 2 als true initialisiert, damit alle Indizes für Testzwecke zur Verfügung stehen. Anschließend beginnt das Sieb mit der Zahl '2' und wenn diese true ist werden alle Multiplikationen von '2' gestrichen. Im nächsten Schritt werden alle gestrichenen Zahlen übersprungen, indem abgefragt wird, ob sie true sind. Wenn die nächste Primzahl gefunden wurde, werden beginnend mit dem Quadrat der Zahl alle Multiplikationen gestrichen. Dieser Vorgang wird wiederholt, bis das Quadrat einer Zahl größer ist als das Maximum. Somit hat man jetzt ein Array, in dem nur die Werte, dessen Index eine Primzahl ist, true sind. Man kann also alle Primzahlen durch eine simple Abfrage in der Konsole ausgeben.

2 Spezielle Primzahlen

In diesem Abschnitt werden sie über zwei der berühmtesten Primzahl arten aufgeklärt zuerst über die Mersenne Primzahlen und anschließend über die Fermatschen Primzahlen. Diese sind extrem wichtig da sie die Rekorde Halten als aktuell größte gefundene Primzahlen.

2.1 Mersenne Primzahlen

Die Mersenne Zahlen werden im allgemeinen als $2^n - 1$ definiert, aber nicht jede von ihnen ist prim. Sie sind so berühmt und geschätzt, da man sie leicht berechnen kann und es einfacher ist, zu überprüfen, ob sie prim sind oder nicht. Da sie eine Wachstumsfunktion als Definition haben, werden sie sehr schnell zu sehr großen Zahlen.

2.1.1 Geschichte der Mersenne Zahlen

Die Mersenne Zahlen sind schon seit mehreren Jahrtausenden bekannt und man glaubte, daß alle Zahlen der Form $2^n - 1$ prim sind. Die erste Zahl, welche dieser Art ist und nicht Prim ist, ist 2047. Dies bewies Hudalcrius Regius im Jahre 1536. Anschließend stellte Pietro Cataldi 1603 die erste Aussage mit den Exponenten 17,19,31 und 37 auf, welche aber von Pierre Fermat im Jahre 1640 korriegiert wurde, daß 23 und 37 nicht prim sind. Der Namensgeber Marin Mersenne fertigte die erste größere Liste an:

$$n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127 \textit{ und } 257$$

im Vorwort zu seiner Cognita Physica-Mathematica im Jahre 1644. Obwohl sie aus heutiger Sicht inkorrekt ist, wurde sein Name der Name dieser Primzahlen. Es dauerte mehrere Jahrhunderte, bis alle Zahlen überprüft wurden und erst 300 Jahre später, im Jahr 1947, war sie dann vollständig bis $n < 258$ überprüft worden. Durch die fortschrittlichen Entwicklungen steigerten sich die Mersenne Primzahlen schnell in den darauffolgenden Jahren. Heutzutage kann jeder die Mersenne Primzahlenfindung unterstützen, indem er seine Rechnerleistung für die "Great Internet Mersenne Prime Search", besser bekannt als GIMPS, über welche sie sich im nachfolgenden Abschnitt genauer informieren können, aufgibt.

2.1.2 GIMPS und PrimeNet

GIMPS ist eine Software, welche eine optimierte Version des Lucas-Lehmer Codes benutzt, um Mersenne Primzahlen zu testen. Sie wurde im Jahr 1996 von George Woltman gegründet und fand mit der 35. Mersenne Primzahl $2^{1398269} - 1$ bereits Ende des Jahres 1996 auch schon ihre erste Primzahl. Anfangs musste der Benutzer die Arbeitsaufträge noch manuell einfordern und so auch die Ergebnisse schicken, bis Scott Kurowski PrimeNet einführte. PrimeNet ermöglicht es, diese Sachen automatisch zu auszuführen und ohne PrimeNet wäre es nicht möglich gewesen, mehrere millionen Arbeitsaufträge auf mehrere tausend Benutzer zu verteilen. Daraus folgte dann ein großes Wachstum des Projekts. Alle Meilensteine von GIMPS können unter `https://www.mersenne.org/report_milestones/` eingesehen werden.

2.1.3 Lucas-Lehmer Test

Der Lucas-Lehmer Test besitzt eine simple Funktionsweise. $Mersenne_p = 2^p - 1$ ist genau dann prim, wenn man $S_0 = 4$ setzt und $S_{n+1} = ((S_n)^2 - 2)$ *modulo* $Mersenne_p$ so lange vollführt, bis S_{p-2} ist und dies durch $Mersenne_p$ teilbar ist.

2.1.4 Programmbeispiel in Java

```
private static boolean Lucas(int p) {
                long wert = 4;
                long prim = (long)Math.pow(2, p)-1;
                System.out.print(prim);
                for (int i = 0; i < p-2; i++) {
                        wert = (wert*wert-2) % prim;
                }
                if (wert == 0) {
                        return true;
                }else {
                return false;
                }
        }
```

2.2 Fermatsche Primzahlen

2.2.1 Definition

Fermatsche Primzahlen sind Fermat-Zahlen, die prim sind. Fermat-Zahlen sind hierbei Zahlen der Form $F_n = 2^{2^n} + 1$, wobei $n \geq 0$.

2.2.2 Informationen

Pierre de Fermat, nach dem die Fermatschen Zahlen benannt sind, behauptete, daß alle Fermatschen Zahlen prim seien. Allerdings fand Leonhard Euler im Jahre 1732 mit der 641 einen Teiler von $F_5 = 4294967297$. Bis heute wurde keine Fermatsche Primzahl mit $n > 5$ gefunden und es wird vermutet, daß es nach den ersten fünf Fermatschen Zahlen keine weiteren Fermatschen Zahlen gibt, die prim sind. Diese Aussage beruht auf dem Primzahlsatz, nach dem die Anzahl der Primzahlen kleiner oder gleich n circa $n/\ln n$ ist. Demnach ist die Wahrscheinlichkeit dafür, daß F_n eine Primzahl ist, etwa $2/\ln_{F_n}$. Die Wahrscheinlichkeit dafür, daß F_6 prim ist, beträgt also $\frac{2}{\ln_{F_6}} = 4.508422\%$ und für F_7 bereits nur noch $\frac{2}{\ln_{F_7}} = 2.254211\%$. Erkennbar ist hierbei, daß sich die Wahrscheinlichkeit für das Prim-Sein von $p_{F_{n+1}}$, immer circa halb so groß ist, wie p_{F_n}. Dies hängt damit zusammen, daß die Fermatsche Zahl F_{n+1} immer etwa doppelt so viele Dezimalstellen wie F_n besitzt.

2.2.3 Rekursive Berechnung von Fermatschen Zahlen

Fermatsche Zahlen können durch Rekursion berechnet werden. Hierfür gibt es verschiedene Algorithmen. Für $n \geq 1$ gilt:

$$F_n = (F_{n-1} - 1)^2 + 1 \tag{1}$$

oder

$$F_n = F_0 \cdot F_1 \cdot ... \cdot F_{n-1} + 2 \tag{2}$$

Hier ist der erste dieser beiden Algorithmen in Programmcode umgesetzt:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double read_input(void)
{
        char buf[256];
        double n = -1;

        printf("Enter n: ");
        scanf("%lf", &n);
        return n;
}

double fermat_1(double num)
{
        double f;
        if (num == 0)
                f = 3;
        else
                f = pow(fermat_1(num - 1) - 1, 2) + 1;
```

```
        printf("F_%.0lf = %.0lf\n", num, f);
        return f;
}

int main(int argc, char *argv[])
{
        double n = read_input();
        fermat_1(n);

        return 0;
}
```

2.2.4 Anwendung Fermatscher Primzahlen in der Geometrie

Carl Friedrich Gauß fand einen Zusammenhang zwischen den Fermatschen Primzahlen und der Konstruktion von regelmäßigen Vielecken. Er bewies, daß ein regelmäßiges Vieleck mit n Seiten nur mit ausschließlich Zirkel und Lineal konstruiert werden kann, wenn n eine Potenz von 2 oder das Produkt einer Protenz von 2 und verschiedenen Fermatschen Primzahlen ist.

2.2.5 Finden neuer Primzahlen

Fermatschen Zahlen sind teilerfremd zueinander. Daraus kann man schließen, daß die Faktoren einer Primfaktorzerlegung sich nicht überschneiden. Da es unendlich viele Fermatsche Zahlen gibt, kann man erschließen, daß es unendlich viele unterschiedliche Primfaktoren und somit unendlich viele Primzahlen gibt.

3 Mathematische Vermutungen zu Primzahlen

Hier kommen wir zu den Unbelegten Vermutungen von Primzahlen. Wir werden versuchen die Probleme mit diesem Vermutungen zu schildern und erklären warum sie nicht belegt sind und welche Probleme auftreten wenn man sie zu belegen versucht.

3.1 Primzahl-Zwillinge

Die perfekten Primzahlzwillinge existieren nur einmal, nämlich die 2 und die 3. Alle darauffolgenden Primzahlzwillinge sind mit einem Abstand von genau 2 zueinander definiert, da alle geraden Zahlen durch zwei teilbar sind.

3.1.1 Eigenschaften von Primzahl-Zwillingen

Die Haupteigenschaft aller Primzahlzwillinge außer 3 und 5 ist, daß sie um eine durch 6 teilbare Zahl gehen, weshalb sie auch als eine Form von $6n-1$ *und* $6n+1$ definiert werden können. Dies hat den Grund, daß jede auf Fünf endende Zahl keine Primzahl ist. Es ist jedoch nicht gewiss, ob es unendlich viele Primzahlzwillinge gibt, obwohl es nach Euklid unendlich viele Primzahlen gibt.

3.1.2 Primzahl-Zwillings Vermutung

Die Primzahl-Zwillings Vermutung besagt, daß es unendlich viele Primzahlen p gibt, sodaß $p+2$ auch prim ist. Dazu gibt es eine verallgemeinerte Version von Alphonse de Polignac, daß für jede natürliche Zahl n auch gilt, es gibt unendlich viele Primzahlen p sodass $p+2k$ auch prim ist, bei $k=1$ ist es dann die Primzahl-Zwillings Vermutung. Da die Wahrscheinlichkeit das eine zahl n Prim ist liegt bei $1/log(n)$ somit könnte man vermuten das $n+2$ auch prim ist wenn es als eigenständige zahl sieht $\pi_2(n) \sim \frac{n}{\log^2(n)}$.

3.1.3 Erste Hardy-Littlewood Vermutung

Die Primzahl-Zwillings Konstante C_2 ist als definiert $C_2 = \prod_{p\geq 3}(\frac{p(p-2)}{(p-1)^2}) \approx 0.66016181584....$ Dazu kann man nach Godfray Harold Hardy und John Edensor Littlewood vermuten, daß $\pi_2(n) \sim 2C_2\frac{n}{\log^2(n)}$. Dieser Vermutung kann zwar zugestimmt werden, aber sie gilt noch als unbewiesen.

3.2 Goldbachsche Vermutung

3.2.1 Definition

Christian Goldbach hat als wahrscheinlich erster vermutet, dass jede gerade Zahl, die groesser als 2 ist, als Summe von zwei Primzahlen dargestellt werden kann. Dies ist die Starke Goldbachsche Vermutung. Aus dieser Starken Vermutung geht die Schwache Goldbachsche Vermutung hervor, welchehingegen aussagt, dass jede ungerade Zahl, welche groesser als 5 ist, als Summe von 3 Primzahlen dargestellt werden kann. Bis heute gibt es keinen Beweis fuer die Goldbachsche Vermutung, verschiedene Computerberechnungen zeigen jedoch, dass die Vermutung bis Zahlengroessen bis zu $4 \cdot 10^{18}$ zutrifft. Auch die Moeglichkeiten, gerade Zahlen groesser als 2 als Summe zweier Primzahlen darzustellen, steigt, je groesser die gerade Zahl ist. Es laesst sich also vermuten, dass die Vermutung wahr ist, bewiesen ist sie dadurch jedoch nicht. Der deutsche Mathemathiker David Hilbert nach die Starke Goldbachsche Vermutung als sein achtes Problem in seine Liste "Hilbertsche Problemeäuf.

3.2.2 Starke Goldbachsche Vermutung

Die Starke Goldbachsche Vermutung, oder auch Binaere Goldbachsche Vermutung genannt, besagt, dass jede gerade Zahl n, die groesser als 2 ist, als Summe zweier Primzahlen abgebildet werden kann. Also $n = p + q$, wobei p und q die beiden Primzahlen sind. Generell ist die Darstellung von n nicht eindeutig. n kann in den meisten Faellen als Summe verschiedener Primzahlpaare dargestellt werden.

3.2.3 Schwache Goldbachsche Vermutung

Die Schwache Goldbachsche Vermutung, oder Ternaere Goldbachsche Vermutung, geht aus der Starken Goldbachschen Vermutung hervor. Sie besagt, dass ungerade Zahlen u, die groesser als 5 sind, als Summe dreier Primzahlen dargestellt werden kann: $u = p + q + r$. Ist der Umstand gegeben, dass die Starke Goldbachsche Vermutung wahr ist, so ist $u = (u - 3) + 3$. $u - 3$ ist hierbei nach Goldbach die Summe zweier Primzahlen, also $u - 3 = p + q$. Somit ist die Darstellung von u als Summe von 3 Primzahlen gegeben: $u = p + q + 3$. Wie bei der Starken Vermutung ist die Abbildung nicht eindeutig. u kann also, mit Ausnahme von der Primzahl 7, durch unterschiedliche Summanden abgebildet werden.

3.2.4 Finden von Summendarstellungen

Das Finden von Summendarstellungen basiert auf dem Sieb des Eratosthenes, welches zuvor schon genauer beschrieben wurde. Beim ersten Sieben eines Siebes der Groesse n, wobei n die gerade Zahl, die groesser als 2 ist dessen Summendarstellung gesucht ist, wird jede Zahl z gestrichen, die nicht prim sind. Beim zweiten Sieben wird anstatt z nun $n - z$ gestrichen. Die sich nun gegenberliegenden Primzahlen bilden jeweils die Summendarstellung der Zahl n.

3.2.5 Beweis

Des oefteren versuchen Amateurmatematiker die Goldbachsche Vermutung zu beweisen und erhalten dabei teils auch die Aufmerksamkeit der Medien. Jedoch bis heute ohne Erfolg. Wenn zwar auch die Vermutung an sich noch nich bewiesen wurde, so gibt es jedoch schon einige interressante Annaeherungen an einen Beweis. Oliver Ramaré bewies im Jahre 1995, dass jede gerade Zahl die Summe von hoechstens sechs Primzahlen ist. Im Jahr 2012 beweist Terence Tao darauf basierend, dass jede ungerade Zahl, welche groesser als eins ist, als Summe von hoechstens 5 Primzahlen dargestellt werden kann. Von der Korrektheit der Vermutung kann

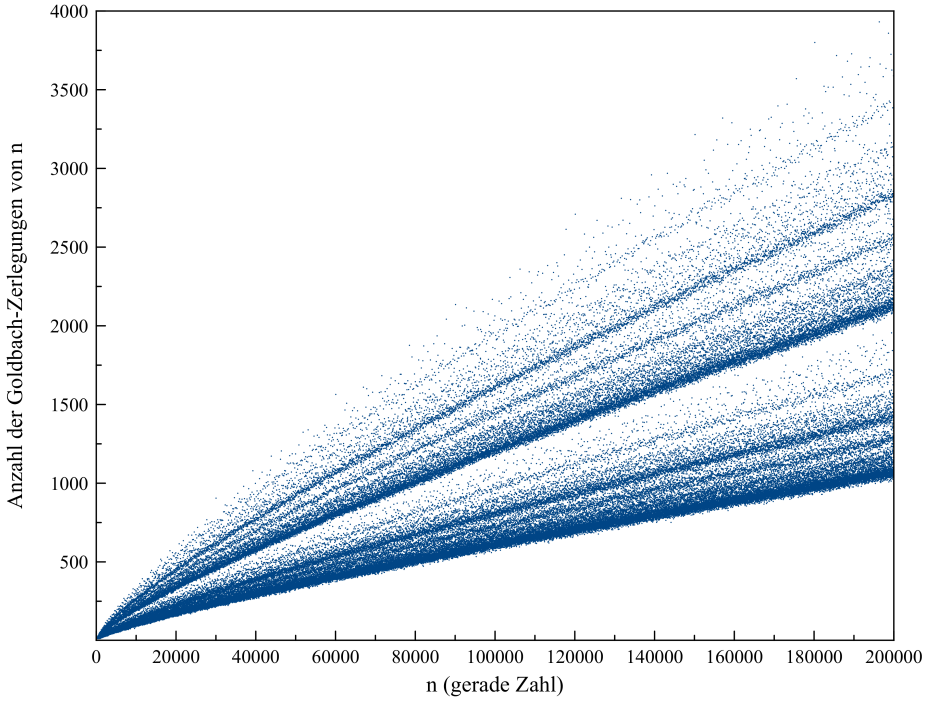

Abbildung 1: Anzahl der Moeglichkeiten gerade Zahlen nach Goldbach als Summe zweier Primzahlen darzustellen

man ebenso ausgehen, da viele Versuche gezeigt haben, dass die Regel alle getesteten Zahlen zutrifft. Das Diagramm in Abbildung 1 zeigt die Anzahl der Moeglichkeiten, eine gerade Zahl als Summe zweier Primzahlen darzustellen. Auf der horizontalen Achse ist hierbei die gerade Zahl n dargestellt, dessen Moeglichkeiten der Goldbach-Zerlegungen man auf der vertikalen Achse ablesen kann. Die Funktion die sich hierbei ergibt, trifft nur bei $n = 0$ den Nullpunkt und steigt wenn n gegen Unendlich geht immer weiter an und trifft dabei nicht mehr die horizontale Achse. Daher kann man ebenfalls davon ausgehen, dass die Goldbachsche Vermutung korrekt ist. Einen Mathematischen Beweis gibt es dennoch nicht.

Quellenverzeichnis

1.