Bachelorarbeit

Darstellung rationaler Zahlen durch Ägyptische Brüche

Eine Untersuchung von Algorithmen und Aufwand

Eingereicht von: Lars Berger
1173278

Aufgabensteller: Prof. Dr. Cornelius Greither

Betreuer: Dr. Soeren Kleine

Abgabedatum: 19.11.2019

Universität der Bundeswehr München
Fakultät für Informatik
Institut für Mathematik und Operations Research

Inhaltsverzeichnis

TODOs

1 Einleitung

Die aktuell ältesten mathematischen Aufzeichnungen der Menschheit sind zwei Papyrusrollen aus dem alten Ägypten, dem sogenannten Golenischev-Papyrus und dem Rhind-Papyrus, jeweils benannt nach ihrem letzten Besitzer, bevor sie in Museen gebracht wurden, die in etwa 4000 Jahre alt sind. Trotz dieses Alters wurden sie erst viel später entdeckt, aus ihren Bruchstücken wieder zusammengesetzt und übersetzt; das Rhind-Papyrus wurde in Teilen während Napoleon Bonapartes Invasion in Ägypten, die 1798 begann, gefunden, konnte aber erst 1922 vervollständigt und anschließend übersetzt werden. Voraussetzung für die Übersetzung dieser Schrift war aber das zuvor gewonnene Wissen über Hieroglyphen und die Demotische Schrift, einem Nachfolger der Hieratischen Schrift, die ihrerseits wieder eine kursive Variante der Hieroglyphen ist, vom sogenannten Stein von Rosette, einem Basalt, in den der gleiche Text in Hieroglyphen, Demotischer Schrift und Griechisch eingraviert ist. Dieser wurde 1799 von Offizieren der Armee Napoleons gefunden und diente danach dem Verständnis der bis dato unverständlichen Schriften des alten Ägypten.[Burton, 2011, S.33 ff]
Das Rhind Papyrus enthält in seiner Präambel folgende Worte, die den Inhalt des Dokuments erfassen sollen: "Ein sorgfältiges Studium aller Dinge, Einblick in Alles, was es gibt, Wissen über alle obskuren Geheimnisse" [1]. In den folgenden 85 Problemen werden dann Multiplikation und Division mittels wiederholter Addition beschrieben sowie Lösungsansätze für Probleme wie das Lösen von linearen Gleichungssystemen in einer Unbekannten.
Diese Arbeit soll sich mit einem sich aus der Division ergebenden Problemfeld befassen, nämlich dem der Ägyptischen Brüche. Zur thematischen Annäherung soll zunächst der Begriff "Ägyptischer Bruch" definiert werden.

Definition 1.0.1. Ein Bruch soll fortan „in ägyptischer Form" bzw. „Ägyptischer Bruch" heißen genau dann, wenn er in der Form

$$\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}, \quad n \in \mathbb{N}, n \geq 1$$

mit paarweise verschiedenen x_i vorliegt.

Diese Art, Brüche zu notieren, wirft unzählige Fragen und Probleme auf, die zum Teil bis heute ungelöst sind. Es bietet sich dadurch ein breites Feld für theoretische und praktische Untersuchungen. Beispielsweise kann das mit Dezimalbrüchen relativ leicht zu erfassende Problem des Vergleichs zweier Brüche nicht auf Ägyptische Brüche übertragen werden, denn gibt es eine mögliche Darstellung für einen Ägyptischen Bruch, gibt es unendlich viele Varianten dafür, wodurch Operationen wie das Vergleichen zweier Ägyptischer Brüche oder das Berechnen von Quadratwurzeln solcher auf systematischem Wege schwierig bis unmöglich sind [Resnikoff und Wells, 1984, S. 62].
In dieser Arbeit sollen verschiedene Fragen zum Thema "Ägyptische Brüche" behandelt und diskutiert werden. Zunächst soll der Ursprung und die Idee dieses Konzepts beleuchtet und begründet werden. Anschließend soll die Betrachtung einiger ausgewählter Algorithmen zur Umwandlung von

[1] [Burton, 2011, S. 37, Übersetzung durch den Autor]

Dezimalbrüchen in Ägyptische Brüche diese vergleichen und untersuchen, wie kompliziert eine solche Umwandlung sein kann, dazu wollen wir für jeden Algorithmus dessen Spezifikationen herausfiltern, die Algorithmen dann mit theoretischen Mitteln gegeneinander bezüglich ihrer Ergebnisse abschätzen und diese Abschätzungen anhand einer strukturellen Untersuchung mittels Computerrechnung überprüfen. Abschließend sollen bereits gefundene Zusammenhänge sowie noch nicht bewiesene Hypothesen aus dem Problemfeld der Ägyptischen Brüche beleuchtet und erklärt werden, um ein möglichst umfangreiches Bild von ebendiesem erfassen zu können.

Anmerkung Die Divergenz der harmonischen Reihe zeigt, dass man mit Brüchen solcher Art durchaus alle rationalen Zahlen $x \in \mathbb{Q}$ erzeugen kann, auch ist es aber möglich, den ganzzahligen Anteil eines gemischten Bruchs gesondert zu betrachten, wodurch der Rest kleiner als 1 ist, weshalb hier auf eine Betrachtung von Brüchen $\frac{a}{b} \in \mathbb{Q}, \frac{a}{b} \geq 1$ verzichtet wird. Zudem werden Brüche $\frac{a}{b} \leq 0$ nicht beachtet, da dies historisch nicht relevant ist: die Ägypter kannten höchstwahrscheinlich keine negativen Zahlen, obwohl sie zumindest ein Symbol für "Nichts" besaßen, also wohl um die Existenz der Null wussten.

2 Ägyptische Arithmetik

Um die Verwendung ägyptischer Brüche historisch zu verstehen, lohnt sich ein kurzer Blick in die Arithmetik des alten Ägypten. Im Folgenden werden dazu die Multiplikation und die darauf aufbauende Division betrachtet, welche die Verwendung von Brüchen bei Division mit nichttrivialem Rest erforderlich macht. Das gesamte arithmetische System der Ägypter baut dabei letztendlich auf der Addition auf.

2.1 Ägyptische Multiplikation

Bei der Multiplikation zweier natürlicher Zahlen $a, b \in \mathbb{N}$ stellt man eine zweispaltige Tabelle auf, die in der linken Spalte die Zweierpotenzen 2^n für die n-te Zeile, mit $n = 0$ beginnend, und rechts das Produkt $a \cdot 2^n$. Zur Multiplikation wählt man nun aus der linken Spalte die Zeilen aus, deren Werte sich zu b addieren, und markiert diese, bspw. mittels eines Hakens (✓). Schließlich werden die Zahlen der rechten Spalte aufaddiert, deren Zeile mittels ✓markiert wurde. Die sich ergebende Summe ist das Ergebnis.

Beispiel 2.1.1. Die Multiplikation $23 \cdot 69$ bzw. $69 \cdot 23$ exemplarisch:

✓	1	69
✓	2	138
✓	4	276
	8	552
✓	16	1104
Summe:	23	1587

✓	1	23
	2	46
✓	4	92
	8	184
	16	368
	32	736
✓	64	1472
Summe:	69	1587

Diese Methode funktioniert, da jede natürliche Zahl $n \in \mathbb{N}$ als Summe paarweise verschiedener Zweierpotenzen darstellbar ist. Es wird im Allgemeinen angezweifelt, dass dies von den Ägyptern je formal bewiesen wurde, aber die Nutzung zeigt, dass sie diesen Zusammenhang zumindest erkannt hatten. [Burton, 2011, S. 38]

2.2 Ägyptische Division

2.2.1 Ganzzahlige Division

Der einfachere Fall der ganzzahligen Division ohne Rest ist dem Prinzip der Multiplikation sehr ähnlich, nur die Herangehensweise ist verändert. Um das Prinzip der Multiplikation anzuwenden zu können, verändert man dafür die Fragestellung. Seien $x \in \mathbb{Q}; a, b \in \mathbb{N}; b \neq 0$. Statt die Lösung für x in der Gleichung $x = \frac{a}{b}$ zu suchen, stellt man die Frage, für welches x gilt $b \cdot x = a$; die Essenz der Frage ist also die gleiche wie die der Multiplikation, nur ist sie umformuliert worden.

Beispiel 2.2.1. Es sei die Division $117 \div 9$ betrachtet. Die Tabelle generiert sich wie oben. Nun wählt man mittels Greedy-Verfahren, also mit jeweils dem größten Wert beginnend, alle Zeilen aus, deren rechte Spalten sich zu 117 addieren, addiert die Einträge in der linken Spalte der ausgewählten Zeilen und erhält das Ergebnis $117 \div 9 = 13$.

✓	1	9
	2	18
✓	4	36
✓	8	72
	$\underline{\underline{13}}$	117

2.2.2 Division mit Rest

Die Division mit Rest $a \div b$ mit $a, b \in \mathbb{N}$ funktioniert ähnlich wie die ganzzahlige, jedoch fügt man an die Tabelle nun noch die Bruchteile von a an, die nötig sind.

Beispiel 2.2.2. Wir betrachten hierfür die Division $117 \div 7$ und stellen die Tabelle auf wie oben.

	1	7
	2	14
	4	28
	8	56
✓	16	112
✗	16	112

Offensichtlich ist das Ergebnis hier noch nicht erreicht, allerdings kann keine weitere Zahl der rechten Spalte ausgewählt werden, ohne 117 zu überschreiten. Folglich sind also kleinere Zahlen als 7 notwendig. Die - aus heutiger Sicht betrachtet - einfache Mathematik des alten Ägypten würde nun, statt die 7 fortlaufend zu verdoppeln, diese zunächst durch sich selbst teilen, um eine 1 zu generieren und dann weiter halbieren, woraus sich diese unvollständige Tabelle ergäbe:

1	7			1	7
$\frac{1}{7}$	1			$\frac{1}{2}$	$3 + \frac{1}{2}$
$\frac{1}{14}$	$\frac{1}{2}$			$\frac{1}{4}$	$1 + \frac{1}{2} + \frac{1}{4}$
$\frac{1}{28}$	$\frac{1}{4}$			$\frac{1}{8}$	$\frac{1}{2} + \frac{1}{4} + \frac{1}{8}$
⋮	⋮			⋮	⋮

Tabelle 1: links: Teilen durch 7, dann fortgesetzte Halbierung von 1; rechts: fortgesetzte Halbierung von 7

Hier wird ein systematisiertes trial-and-error-Verfahren verwendet. Da sich aus Tabelle 1 (links) die Harmonische Reihe ergibt, aber ein Gesamtwert von $117 - 112 = 5$ benötigt wird, fällt die Wahl

zunächst auf das größte Element in Tabelle 1 (rechts) mit Wert $3 + \frac{1}{2}$. Es folgt nun ein Rest von $5 - (3 + \frac{1}{2}) = 1 + \frac{1}{2}$, welcher durch die Tabelle 1 (links) mit den Werten 1 und $\frac{1}{2}$ genau erfüllt wird. Es folgt die Gesamttabelle:

	1	7
	2	14
	4	28
	8	56
✓	16	112
✓	$\frac{1}{2}$	$3 + \frac{1}{2}$
✓	$\frac{1}{7}$	1
✓	$\frac{1}{14}$	$\frac{1}{2}$
	$16 + \frac{1}{2} + \frac{1}{7} + \frac{1}{14}$	117

Tabelle 2: Die vollständige Divisionstabelle

Sei $x \in \mathbb{N}$ und n der Divisor der gewählten Division. Typische Brüche, die in der linken Spalte verwendet wurden, weil diese einfach zu berechnen sind, waren $\frac{1}{2^x}$, sowie $\frac{1}{n \cdot 2^x}$.

Aus dieser Methodik heraus ergibt sich die Notation der Ägyptischen Brüche. Im Folgenden soll die Frage geklärt werden, wie man gemeine Brüche in Brüche ägyptischer Form umwandeln kann und wie das im alten Ägypten umgesetzt wurde.

2.3 Ermittlung Ägyptischer Zerlegungen von Brüchen

Die Ägypter brauchten nun also ein System, mit dessen Hilfe sie die Zerlegung von Brüchen in eine Summe von Stammbrüchen mit paarweise verschiedenen Nennern berechnen konnten. Die einfache Zerlegung

$$\frac{a}{b} = \underbrace{\frac{1}{b} + \frac{1}{b} + ... + \frac{1}{b}}_{a-mal}$$

kam dabei nicht in Frage, da die Ägypter es als „unnatürlich" ansahen, dass es mehr als diesen einen, wahren Teiler $\frac{1}{b}$ einer Zahl geben sollte. [Burton, 2011, S.39]
Für die Brüche $\frac{2}{n}$ für $n \in \mathbb{N}$ ungerade und $5 \leq n \leq 101$ findet sich dafür im Rhind-Papyrus eine Tabelle mit der jeweiligen Zerlegung. Auch waren einige Regeln bekannt, beispielsweise

$$\frac{2}{n} = \frac{1}{n} + \frac{1}{2n} + \frac{1}{3n} + \frac{1}{6n}.$$

Tatsächlich wurde diese Regel in der zuvor genannten Tabelle des Papyrus nur einmal, bei $\frac{2}{101} = \frac{1}{101} + \frac{1}{202} + \frac{1}{303} + \frac{1}{606}$, verwendet, sonst wurden kürzere Zerlegungen gewählt. Trotz immenser Bemühungen ist es bisher nicht gelungen, das System zu ermitteln, mittels welchem diese Tabelle zustande kam. [Burton, 2011, S. 41]
Im Zuge der modernen Untersuchungen Ägyptischer Brüche sind aber noch viele weitere Methoden

zur Zerlegung jeglicher Dezimalbrüche in Brüche Ägyptische Form gefunden worden, eine Auswahl solcher Zerlegungsalgorithmen sollen im Folgenden aufgezeigt, untersucht und verglichen werden.

3 Algorithmen zur Erstellung Ägyptischer Brüche

Im Folgenden sollen verschiedene Algorithmen erklärt und verglichen werden, mit welchen sich rationale Brüche in Ägyptische Brüche gemäß Definition 1.0.1 zerlegen lassen. Die dabei aufgezeigten Methoden wurden über mehrere Jahrhunderte hinweg entwickelt und weisen dementsprechend signifikante Unterschiede auf. Im Anschluss an die Erklärung der Algorithmen soll ein Vergleich gezogen werden, der die Effizienz anhand der Kriterien "Anzahl der verwendeten Stammbrüche" und "Länge des größten Nenners" vergleicht. Die nun betrachteten Algorithmen werden sein:

- der Fibonacci-Sylvester-Algorithmus (auch: Greedy-Algorithmus)
- der Farey-Folgen-Algorithmus
- der Kettenbruch-Algorithmus

Da im alten Ägypten noch keine negativen Zahlen bekannt waren, beschränken sich entsprechend die Algorithmen auf die positiven rationalen Brüche.

Definition 3.0.1. Da wir im Folgenden nur positive rationale Brüche zwischen 0 und 1 betrachten wollen, sei die Menge $\mathbb{Q}_+$ wie folgt definiert:

$$\mathbb{Q}_+ := \{x \in \mathbb{Q} : 0 < x < 1\}.$$

3.1 Der Greedy-Algorithmus

Sei $\frac{p}{q} \in \mathbb{Q}_+$. Eine schon seit dem 13. Jahrhundert bekannte Methode, einen Ägyptischen Bruch für rationale Brüche $\frac{p}{q}$ zu finden, ist der Greedy-Algorithmus. Dieser wurde zuerst 1202 von Fibonacci entwickelt und 1880 von James Joseph Sylvester wiederentdeckt und weiterentwickelt, weswegen er auch den Namen "Fibonacci-Sylvester-Algorithmus" trägt [Burton, 2011, S.44f]. Dabei werden solange jeweils die größtmöglichen Stammbrüche $\frac{1}{x_i}$ gesucht, sodass

$$\frac{1}{x_i} \leq \frac{p}{q} - \sum_{j=1}^{i-1} \frac{1}{x_j} < \frac{1}{x_i - 1}, \tag{1}$$

wobei gilt, dass

$$x_j \neq x_k, \forall j \neq k;\ j,k \in \{1,..,i\},$$

bis

$$\frac{a}{b} = \frac{1}{x_1} + \frac{1}{x_2} + ... + \frac{1}{x_i} = \sum_{j=1}^{i} \frac{1}{x_j}.$$

Als Anweisungsfolge lässt sich der Algorithmus folgendermaßen formulieren.

Algorithmus 3.1.1. Der Greedy-Algorithmus.

1. finde den größten, noch nicht verwendeten Stammbruch $\frac{1}{x}$, sodass $\frac{1}{x} \leq \frac{p}{q}$.

2. setze $\frac{1}{x}$ als weiteren Summanden des Ergebnisses
3. falls $\frac{p}{q} - \frac{1}{x} > 0$, gehe zu Schritt 1 mit $\left(\frac{p}{q}\right) \leftarrow \left(\frac{p}{q} - \frac{1}{x}\right)$.

Da in jedem Fall der größtmögliche, noch nicht vorhandene Bruch gesucht wird, der noch in die Summe der Stammbrüche passt, ohne dass diese zu groß wird, kann es zu sehr ungünstigen Ergebnissen mit extrem langen Nennern kommen; ein anschauliches Beispiel dafür ist:

$$\frac{5}{121} = \frac{1}{25} + \frac{1}{757} + \frac{1}{763.309} + \frac{1}{873.960.180.913} + \frac{1}{1.527.612.795.642.093.418.846.225},$$

wobei man den Bruch auch folgendermaßen zerlegen kann:

$$\frac{5}{121} = \frac{1}{33} + \frac{1}{121} + \frac{1}{363}.$$

Aufgrund dieser Umstände scheint es unsinnig, den Greedy-Algorithmus zu verwenden; immerhin lässt sich beweisen, dass dieser immer terminiert. Im Anhang 6.2 findet sich eine eigene Implementierung des Greedy-Algorithmus.

Satz 3.1.2. *Der Greedy-Algorithmus, wie oben beschrieben, terminiert für jede Eingabe.*

Beweis. Zur Verkürzung der Schreibweise sei $x = x_1$. Für die erste Iteration des Greedy-Algorithmus ergibt sich aus Gleichung (1):

$$\frac{1}{x} \leq \frac{p}{q} < \frac{1}{x-1}. \tag{2}$$

Daraus folgt ein Rest r von

$$r = \frac{p}{q} - \frac{1}{x} = \frac{px - q}{qx},$$

der den Zähler $(px - q)$ hat, welcher kleiner als p ist. Dies folgt aus (2):

$$\begin{aligned} \frac{p}{q} < \frac{1}{x-1} \quad &\Leftrightarrow \quad p(x-1) < q \\ &\Leftrightarrow \quad px - p < q \\ &\Leftrightarrow \quad px - q < p. \end{aligned}$$

Somit verkleinert sich der Rest r mit jedem Schritt und erreicht nach endlich vielen Schritten Null, wie gefordert. □

Abschätzung Sei $\frac{p}{q} \in \mathbb{Q}_+$ der zu zerlegende Bruch. Der Greedy-Algorithmus liefert einer Zerlegung mit maximal q Termen. [Gong, 1992, S.2]

3.2 Der Farey-Folgen-Algorithmus

Eine weitere Methode zur Erstellung Ägyptischer Brüche stellt der von M. N. Bleicher vorgeschlagene Farey-Folgen-Algorithmus dar, der seinen Namen aus dem Umstand bezieht, dass die Farey-Folge dafür genutzt wird. [Guy, 1981, S. 88]

Definition 3.2.1. Sei $q \in \mathbb{N}$. Die Farey-Folge der Ordnung q, F_q, ist definiert als die aufsteigend sortierte Folge aller einmalig darin vorkommenden gekürzten Brüche $\frac{a}{b} \in \mathbb{Q}$, für die gilt: $0 \leq a \leq b \leq q$, $b \neq 0$.

Beispiel 3.2.2. Sei $q = 5$. Die Farey-Folge der Ordnung 5 ist

$$F_5 = \left\{ \frac{0}{1}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{1}{1} \right\}.$$

Der Algorithmus funktioniert dann wie folgt.

Algorithmus 3.2.3. Sei $\frac{p}{q} \in \mathbb{Q}_+$ in reduzierter Form der zu zerlegende Bruch.

1. Konstruiere F_q.
2. Sei $\frac{r}{s}$ der zu $\frac{p}{q}$ adjazente Bruch in F_q, sodass $\frac{r}{s} < \frac{p}{q}$. Aufgrund der Eigenschaften der Farey-Folge gilt dann
$$\frac{p}{q} = \frac{1}{qs} + \frac{r}{s},$$
wobei $s < q$, $r < p$ [Beck, 2000, S. 425].
3. Wiederhole dieses Vorgehen für $\frac{r}{s}$ solange, bis $s = 1 \Leftrightarrow r = 0$.

Satz 3.2.4. *Algorithmus 3.2.3 terminiert für jede Eingabe.*

Beweis. Sei $\frac{p}{q} \in \mathbb{Q}_+$ der zu zerlegende rationale Bruch, $\frac{r}{s}, \frac{t}{u} \in \mathbb{Q}_+$ die zu $\frac{p}{q}$ in F_q adjazenten Brüche, wobei gilt $\frac{r}{s} < \frac{p}{q} < \frac{t}{u}$.
Es gilt

$$\frac{p}{q} = \frac{1}{qs} + \frac{r}{s}.$$

Falls $r = 0$, dann $\frac{p}{q} = \frac{1}{qs}$ und der Algorithmus terminiert. Sonst wird der Algorithmus für $\frac{r}{s}$ in F_s wiederholt. Es gilt $s < q$, da kein Nenner in F_q größer als q ist und zwei beliebige Brüche mit demselben Nenner niemals adjazent zueinander sind. Somit wird nach endlich vielen Schritten $r = 0$ erreicht. Da in jeder Farey-Folge der einzige Bruch mit Zähler Null $\frac{0}{1}$ ist, ist dies der Abbruchfall. □

Abschätzung Sei $\frac{p}{q} \in \mathbb{Q}_+$ der zu zerlegende Bruch. Der Farey-Folgen-Algorithmus liefert dafür eine Zerlegung mit maximal p Termen und einem größten Nenner von höchstens $q(q-1)$. [Bleicher, 1972, S.343]

Da die Farey-Folge F_q schon für mäßig große q sehr groß wird, bietet sich für das tatsächliche Berechnen eine Optimierung an, indem nur der relevante Teil der Farey-Folge konstruiert wird. Anwendung findet dabei das Bisektionsverfahren.

Definition 3.2.5. Die im Folgenden verwendete Mediante zweier rationaler Brüche $\frac{a}{b}, \frac{c}{d} \in \mathbb{Q}_+$ $mediant : \mathbb{Q}_+ \times \mathbb{Q}_+ \to \mathbb{Q}_+$ sei definiert als:

$$mediant\left(\frac{a}{b}, \frac{c}{d}\right) = \frac{a+c}{b+d}.$$

Beispiel 3.2.6. Sei $\frac{p}{q} = \frac{21}{23}$. Die obere bzw. untere Schranke ist in unserem Fall 0 bzw. 1. Die Mediante liegt also bei $\frac{1}{2}$, wir stellen fest: $\frac{1}{2} < \frac{21}{23} < 1$, also setzen wir die Suche im Intervall $[\frac{1}{2}, 1]$ fort. Die Mediante liegt nun bei $\frac{2}{3}$, $\frac{2}{3} < \frac{21}{23} < 1$ usw. Bei der Mediante $\frac{11}{12}$ stellen wir fest: $\frac{10}{11} < \frac{21}{23} < \frac{11}{12}$. Daraus folgt der relevante Teil von F_{23}, hier F_{23rel} genannt:

$$F_{23rel} = \left\{\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{7}{8}, \frac{8}{9}, \frac{9}{10}, \frac{10}{11}, \frac{21}{23}, \frac{11}{12}\right\}.$$

Weil für die Adjazenzberechnung von $\frac{p}{q}$ nur die unmittelbare Umgebung nötig ist, fallen alle Brüche, die größer als der größere der zu $\frac{p}{q}$ adjazenten Brüche sind, aus der gekürzten Farey-Folge heraus; im Beispiel betrifft dies $\frac{1}{1}$. Somit enthält die gekürzte Farey-Folge nur noch 12 der andererseits 173 zu berechnenden Elemente von F_{23}.

3.3 Binäralgorithmus

Dass die Ägypter schon damals implizit eine Art Binärschreibweise für ihre Multiplikation und Division verwendeten, wie in Abschnitt 1 erläutert, ist inzwischen bekannt. Nun kann man den gleichen Umstand auch für einen Konstruktionsalgorithmus verwenden, wie im Folgenden gezeigt wird.
Seien $m, n \in \mathbb{N}$ und $N = 2^n$. Alle $m < N$ lassen sich als Summe paarweise verschiedener Teiler von N beschreiben, folglich also mit maximal n Summanden. Praktisch lässt sich dies am Einfachsten anhand der Binärdarstellung von m erkennen.

Algorithmus 3.3.1. Sei $\frac{p}{q} \in Q_+$, $\frac{p}{q} < 1$ in gekürzter Form und $k \in \mathbb{N}$.

1. Finde $N_{k-1} < q \leq N_k$ wobei $N_k = 2^k$ ist.
2. Falls $q = N_k$, schreibe p als Summe von Teilern von N_k, hier d_i genannt:

$$\frac{p}{q} = \sum_{i=1}^{j} \frac{d_i}{N_k} = \sum_{i=1}^{j} \frac{1}{\frac{N_k}{d_i}}$$

3. Sonst seien $s, r \in \mathbb{N}, 0 \leq r < q$ so gewählt, dass:

$$pN_k = qs + r.$$

Es folgt:

$$\frac{p}{q} = \frac{pN_k}{qN_k} = \frac{qs+r}{qN_k} = \frac{s}{N_k} + \frac{r}{qN_k}.$$

4. Schreibe $s = \sum d_i$ und $r = \sum d_i'$, wobei d_i, d_i' jeweils paarweise verschiedene Teiler von N_k sind.

5. Erhalte den Ägyptischen Bruch:

$$\sum \frac{1}{\frac{N_k}{d_i}} + \sum \frac{1}{\frac{qN_k}{d_i'}}$$

Satz 3.3.2. *Der Binäralgorithmus, wie in 3.3.1 beschrieben, terminiert für jede Eingabe.*

Beweis. Falls $q = N_k$, hat das Ergebnis offensichtlich maximal k Terme, da sich N_k als Summe seiner $\log_2 N = k$ Terme schreiben lässt; in der Summe sind die d_i paarweise verschieden, somit auch die $\frac{N_k}{d_i}$ und es gibt keinen Term mehrfach.
Falls $q < N_k$, gilt

$$qs + r = pN_k < qN_k.$$

Somit gibt es eine Zerlegung in Ägyptische Brüche jeweils für s und r. Diese beiden Zerlegungen liefern für sich genommen nach dem Argument aus Fall "$q = N_k$" paarweise verschiedene Stammbrüche. Dass diese sogar zusammengenommen paarweise verschieden sind, folgt daraus, dass die zu s gehörenden Nenner immer Zweierpotenzen sind, die zu r gehörigen aber niemals. □

Abschätzung Sei $\frac{p}{q} \in \mathbb{Q}_+, \frac{p}{q} < 1$. Gong zeigte, dass die Komplexitätsklasse für die Anzahl der Terme $O(\log n)$ ist [Gong, 1992, S. 12].
Falls $q = N_k$ ist, folgt, dass der maximale Nenner q ist. Anderenfalls definiert sich dieser aus dem Term $\frac{r}{qN_k}$ mit einer 1 in der Zerlegung von r in Teiler von N_k. Schlimmstenfalls ist q dann der Nachfolger einer Zweierpotenz und es gilt

$$N_k = 2(q-1).$$

Daraus folgt als kleinstmöglicher Bruch $\frac{1}{q \cdot 2(q-1)} = \frac{1}{2(q^2-q)}$, was die obere Schranke für den größten Nenner $2(q^2 - q)$ impliziert, die in der Komplexitätsklasse $O(q^2)$ enthalten ist.

Da es für diesen Algorithmus zwei Fälle gibt, soll für jeden Fall ein Beispiel gezeigt werden. Dafür beginnen wir mit dem einfachen Fall.

Beispiel 3.3.3. Sei $\frac{9}{16}$ der zu zerlegende Bruch. $N_k = 16$, da $8 < 16 \leq 16$.
Da $16 = 16$, wird nach Schritt (2) aus Algorithmus 3.3.1 verfahren:

$$\frac{9}{16} = \frac{8+1}{16} = \frac{1}{2} + \frac{1}{16}.$$

Ist der Nenner des zu zerlegenden Bruchs also eine Zweierpotenz, terminiert der Algorithmus sehr schnell. Anders ist dies, sollte es sich um keine Zweierpotenz handeln, wie das nächste Beispiel zeigt.

Beispiel 3.3.4. Sei $\frac{21}{23}$ der zu zerlegende Bruch. $N_k = 32$, da $16 < 23 \leq 32$.
Da $23 < 32$ ist, wird nach Schritt (3) aus Algorithmus 3.3.1 verfahren:

$$\frac{21}{23} = \frac{21 \cdot 32}{23 \cdot 32}.$$

Aus der Bedingung $0 < r < 32$ folgt $s = 29$ und $r = 5$, da

$$qs + r = 29 \cdot 23 + 5 = 21 \cdot 32 = pN_k.$$

Daraus folgt:

$$\frac{21}{23} = \frac{23 \cdot 29 + 5}{23 \cdot 32} = \frac{29}{32} + \frac{5}{23 \cdot 32}.$$

Aus

$$\frac{29}{32} = \frac{16 + 8 + 4 + 1}{32} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{32}$$

und

$$\frac{5}{23 \cdot 32} = \frac{4 + 1}{23 \cdot 32} = \frac{1}{23 \cdot 8} + \frac{1}{23 \cdot 32}$$

folgt:

$$\frac{21}{23} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} + \frac{1}{184} + \frac{1}{736}.$$

3.4 Beispielrechnungen

Anhand von drei Beispielen soll die Rechnung der einzelnen Methoden zur besseren Vergleichbarkeit verdeutlicht werden. Am Ende dieses Abschnitts sind die Ergebnisse tabellarisch gegenübergestellt.

Beispiel 3.4.1. Wir wollen zunächst einen einfachen Bruch zerlegen, für den alle drei der aufgezeigten Algorithmen das gleiche Ergebnis liefern, auch wenn sie dieses auf unterschiedlichem Wege erreichen. Sei $\frac{5}{9}$ der betrachtete Bruch.

Greedy-Algorithmus Es wird der größte Stammbruch $\frac{1}{x_1} \in \mathbb{Q}_+$ gesucht mit $\frac{1}{x_1} \leq \frac{5}{9} < \frac{1}{x_1 - 1}$. Es folgt

$$\frac{1}{x_1} = \frac{1}{2}, \text{ da } \frac{1}{2} \leq \frac{5}{9} < \frac{1}{1}.$$

Der verbleibende Rest r ist dann

$$r = \frac{5}{9} - \frac{1}{2} = \frac{1}{18}$$

und der Algorithmus terminiert, da die Summe dieser beiden Stammbrüche genau $\frac{5}{9}$ entspricht.

Farey-Folgen-Algorithmus Wir bilden den notwendigen Teil der Farey-Folge mit Ordnung 9:

$$F_{9rel} = \left\{\frac{0}{1}, \frac{1}{2}, \frac{5}{9}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{1}{1}\right\},$$

woraus folgt, dass $\frac{1}{2}$ der zu $\frac{5}{9}$ adjazente Bruch in F_9 ist. Daraus folgt

$$\frac{5}{9} = \frac{1}{9 \cdot 2} + \frac{1}{2} = \frac{1}{2} + \frac{1}{18}$$

und der Algorithmus terminiert nach dem ersten Schritt, da der Rest $\frac{r}{s} = \frac{1}{2}$ schon Stammbruch ist.

Binäralgorithmus Die kleinste Zweierpotenz, die größer als der Nenner 9 ist, ist 16, da $8 < 9 < 16$; somit ist $N_k = 16$. Da der Nenner keine Zweierpotenz ist, springen wir zu Schritt 3 des Algorithmus 3.3.1 und schreiben:

$$\frac{5}{9} = \frac{5 \cdot 16}{9 \cdot 16} = \frac{9 \cdot 8 + 8}{9 \cdot 16} = \frac{8}{16} + \frac{8}{144} = \frac{1}{2} + \frac{1}{18}.$$

Es ergibt sich also bei allen Algorithmen das gleiche Ergebnis, obwohl die Herangehensweise sehr unterschiedlich ist. Das ist aber nur ein Ausnahmefall, die folgenden Beispiele werden zeigen, wie unterschiedlich die Ergebnisse werden können.

Algorithmus	Anzahl der Terme	Größter Nenner	Zerlegung
Greedy	2	18	$\frac{1}{2} + \frac{1}{18}$
Farey-Folgen-Algorithmus	2	18	$\frac{1}{2} + \frac{1}{18}$
Binär-Algorithmus	2	18	$\frac{1}{2} + \frac{1}{18}$

Tabelle 3: Die Zerlegung von $\frac{5}{9}$ im Vergleich

Beispiel 3.4.2. Sei $\frac{24}{31}$ der zu zerlegende Bruch. Um Zeit zu sparen, werden hier die Lösungswege nur noch angeschnitten.

Greedy-Algorithmus Wieder suchen wir iterativ die größten Stammbrüche, bis diese in ihrer Summe $\frac{24}{31}$ ergeben. Nach 4 Schritten erhält man das Ergebnis:

$$\frac{24}{31} = \frac{1}{2} + \frac{1}{4} + \frac{1}{42} + \frac{1}{2604}.$$

Farey-Folgen-Algorithmus Wie bisher wird zunächst der zu $\frac{24}{31}$ in F_{31} adjazente Bruch gesucht, mit dem nach Rechenvorschrift fortgefahren wird, sodass sich nach der ersten Iteration

$$\frac{24}{31} = \frac{1}{31 \cdot 22} + \frac{17}{22} = \frac{1}{682} + \frac{17}{22}$$

ergibt. Mit den folgenden 5 Iterationen ergibt sich:

$$\frac{24}{31} = \frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \frac{1}{52} + \frac{1}{286} + \frac{1}{682}.$$

Binäralgorithmus Schritt 1 liefert $N_k = 32$. Schritt 3 zufolge gilt

$$\frac{24}{31} = \frac{24 \cdot 31 + 24}{31 \cdot 32} = \frac{16+8}{32} + \frac{16+8}{31 \cdot 32} = \frac{1}{2} + \frac{1}{4} + \frac{1}{62} + \frac{1}{124}.$$

Die Ergebnisse, die in Tabelle 4 aufgelistet sind, zeigen einige signifikante Unterschiede: Vergleicht man den Greedy- mit dem Farey-Folgen-Algorithmus, wird deutlich, dass durch die Erhöhung der Anzahl der Terme von 4 auf 6 der größte vorkommende Nenner um mehr als den Faktor 3 verringert werden kann. Es geht aber offensichtlich noch besser, denn der Binäralgorithmus liefert nur 4 Terme, deren größter Nenner aber um den Faktor 21 kleiner ist als der des Greedy-Algorithmus und um den Faktor 5,5 kleiner als der des Farey-Folgen-Algorithmus. Ersteres ist allein der Habgier[2] des Greedy-Algorithmus geschuldet, da er an dritter Stelle statt der besseren Wahl $\frac{1}{62}$ den größeren Bruch $\frac{1}{42}$ wählt und somit den Rest so stark verkleinert, dass dieser nur durch einen relativ großen Nenner ausgedrückt werden kann. Hier ist also der Binäralgorithmus den anderen beiden deutlich überlegen.

Algorithmus	Anzahl der Terme	Größter Nenner	Zerlegung
Greedy	4	2604	$\frac{1}{2} + \frac{1}{4} + \frac{1}{42} + \frac{1}{2604}$
Farey-Folgen-Algorithmus	6	682	$\frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \frac{1}{52} + \frac{1}{286} + \frac{1}{682}$
Binär-Algorithmus	4	124	$\frac{1}{2} + \frac{1}{4} + \frac{1}{62} + \frac{1}{124}$

Tabelle 4: Die Zerlegung von $\frac{24}{31}$ im Vergleich

Beispiel 3.4.3. Dass der Greedy-Algorithmus nicht grundsätzlich der schlechteste ist, zeigt das Beispiel $\frac{12}{17}$.

Der Greedy-Algorithmus liefert

$$\frac{12}{17} = \frac{1}{2} + \frac{1}{5} + \frac{1}{170}.$$

[2] greedy, engl. für: habgierig, gierig, gefräßig

Der Farey-Folgen-Algorithmus liefert

$$\frac{12}{17} = \frac{1}{2} + \frac{1}{6} + \frac{1}{30} + \frac{1}{170}.$$

Der Binäralgorithmus liefert

$$\frac{12}{17} = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{68} + \frac{1}{272}.$$

Wie die Zusammenfassung in Tabelle 5 zeigt, ist in diesem Beispiel der Greedy-Algorithmus leicht überlegen. An den Zweierpotenzen der ersten Terme des Binäralgorithmus lässt sich sehr gut dessen Natur kennen, die ihm aber in diesem Beispiel nicht zu sonderlicher Effizienz verhilft.

Algorithmus	Anzahl der Terme	Größter Nenner	Zerlegung
Greedy	3	170	$\frac{1}{2} + \frac{1}{5} + \frac{1}{170}$
Farey-Folgen-Algorithmus	4	170	$\frac{1}{2} + \frac{1}{6} + \frac{1}{30} + \frac{1}{170}$
Binär-Algorithmus	5	272	$\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{68} + \frac{1}{272}$

Tabelle 5: Die Zerlegung von $\frac{12}{17}$ im Vergleich

Es zeigt sich also, dass sich anhand relativ weniger Beispiele nicht sagen lässt, ob ein Algorithmus besser ist als ein anderer. Dazu braucht es einen großen, systematisch aufgebauten Datensatz, der dann statistisch ausgewertet wird. Trotzdem lässt sich daraus nicht ableiten, welcher Algorithmus im Einzelfall besser funktioniert. Viele Algorithmen sind mit aufwändigen Beweisen ihrer Schranken bezüglich "Länge des größten Nenners" und "Anzahl der entstehenden Terme" verbunden, diese sagen aber selten etwas über die durchschnittlichen Ergebnisse aus.

3.5 Fazit und Vergleich

Wie in den vorangegangenen Beispielen zu sehen, liefern verschiedene Methoden z.T. stark voneinander abweichende Ergebnisse. Wie optimal zwei Ergebnisse im Vergleich zueinander sind, wird oft in den Maßeinheiten der oberen Schranken der "Länge des größten Nenners" bzw. der "Anzahl der Summanden" angegeben. Sei $\frac{p}{q} \in \mathbb{Q}_+$ der untersuchte Bruch. In Tabelle 7 sind die entsprechenden Werte zum Vergleich aufgelistet.[Bleicher, 1972, S. 343]

Algorithmus	Anzahl der Summanden	Größtmöglicher Nenner
Fibonacci-Sylvester(Greedy)	p	-
Farey-Folgen-Algorithmus	p	$q(q-1)$
Binäralgorithmus	$O(\log q)$	$2(q^2 - q)$

Tabelle 7: Vergleich der beschriebenen Algorithmen (obere Schranken)

Für den größtmöglichen Nenner des Greedy-Algorithmus gibt es eine Besonderheit, denn die Nenner dort wachsen exponentiell, weshalb sich keine genaue obere Schranke angeben lässt [Bleicher und Erdös, 1976, S. 157].

4 Auswertung einiger Testreihen

Zur effizienten Untersuchung und Rechnung zahlreicher Beispiele wurden die in Kapitel 3 aufgezeigten Algorithmen in PARI/GP umgesetzt. (The PARI Group [2018])
Alle Beispiele wurden dabei ausschließlich mit dem in Abschnitt 6 gelisteten Programmcode bzw. mithilfe der darin aufgerufenen, nativ in PARI/GP enthaltenen Funktionen gerechnet. Visualisierungen wurden mit *pyplot* aus dem *matplotlib 3.1.1*-Paket für Python 3.7 erstellt. Es sei darauf hingewiesen, dass einige der folgenden Grafiken mit logarithmischer, andere mit linearer Skala dargestellt wurden.

4.1 Methodik

Zur Testung der genannten Algorithmen wurde die folgende Methodik umgesetzt. Alle drei Algorithmen durchliefen die Tests zwecks Vergleichbarkeit auf sehr ähnlichen Rechnern mit gleichen Bedingungen, die Laufzeiten sind somit vergleichbar. Getestet wurde ausgehend von $n \in \mathbb{N}$ mit $3 \leq n \leq 10.000$. Der Datensatz für ein gewähltes n definiert sich durch die Menge der betrachteten Brüche $M_n = \{\frac{x}{n} \,|\, 1 \leq x < n \wedge \mathrm{ggT}(x, n) = 1\}$ und enthält neben dem n selbst folgende Eigenschaften, die zugleich den nachgestellten Namen bemkommen:

- die durchschnittliche Anzahl der Summanden, *avgTerms(n)*
- das Minimum der Anzahl der Summanden, *minTerms(n)*
- das Maximum der Anzahl der Summanden, *maxTerms(n)*
- das Minimum des jeweils größten Nenners, *minDenom(n)*
- das Maximum des jeweils größten Nenners, *maxDenom(n)*.

Wird das entsprechende Kriterium nur für einen einzelnen Algorithmus betrachtet, ist dies am Index ersichtlich, der Datenpunkt n der maximalen Anzahl der Terme für den Farey-Folgen-Algorithmus wird beispielsweise durch $maxTerms_{farey}(n)$ angegeben. Somit entstanden 10.000 Datensätze pro Algorithmus mit jeweils 5 nutzbaren Parametern, insgesamt also 150.000 Datenpunkte, die im Folgenden ausgewertet werden sollen.

4.2 Anfängliche Probleme und Effizienzsteigerung während der Implementierung

In den ersten Testläufen traten einige Unannehmlichkeiten auf, da Teile des Codes in der direkten Umsetzung der formalen Beschreibung nicht immer optimal liefen.

Suche nach Nennern im Greedy-Algorithmus Beispielsweise gab es im ursprünglichen Test des Greedy-Algorithmus den Fall der Zerlegung von $\frac{5}{121}$, dessen Berechnung mittels des Funktionsaufrufs *greedy(5/121)* nach ca. 7 Stunden ohne Ergebnis abgebrochen wurde. Grund hierfür ist das Problem, den größten Stammbruch $\frac{1}{x} \in \mathbb{Q}_+$ zu finden, der kleiner als ein gegebener rationaler Bruch $\frac{p}{q} \in \mathbb{Q}_+$ ist. In der ersten Implementierung wurde dafür in der Folge $(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, ..., \frac{1}{x})$ gesucht, was für große Nenner offensichtlich sehr lange dauert. In einer Optimierung wurde folgende Vorgehensweise umgesetzt, um die Effizienz signifikant zu steigern. Konstruiere die Folge $(\frac{1}{2^1}, \frac{1}{2^2}, ..., \frac{1}{2^k})$ mit $2^k < \frac{p}{q} < 2^{k+1}$. Wähle dann aus dieser Folge mit dem größten Nenner beginnend Elemente aus, die in ihrer Summe kleiner oder gleich $\frac{p}{q}$ sind. Das Ergebnis ist eine Summe von Kehrwerten der gewählten Zweierpotenzen, hier $\frac{1}{x}$ genannt, für die gilt: $\frac{1}{x} < \frac{p}{q} < \frac{1}{x-1}$. Somit ist $\frac{1}{x}$ der größte Stammbruch, der kleiner als $\frac{p}{q}$ ist, wurde aber mit wesentlich weniger Rechenaufwand gefunden. Dadurch steigerte sich die Effizienz des Greedy-Algorithmus enorm: Das genannte Beispiel $\frac{5}{121}$ wurde in unter einer Millisekunde korrekt berechnet, zahlreiche weitere Beispiele zeigten ähnliche Auswirkungen. Deshalb trägt diese Implementierung auch den Namen *greedy_fast(fraction)*.

Konstruktion der Farey-Folgen Ein weiteres Problem stellte die Berechnung der Farey-Folgen dar. Einerseits haben solche Folgen F_q selbst mit mäßig großem q schon sehr viele Elemente, so ist z. B. $|F_{280}| = 23.861$. Aus den Eigenschaften der Farey-Folgen kann man sehen, dass diese mit steigender Ordnung exponentiell an Größe gewinnen. Andererseits werden die Zähler und Nenner der enthaltenen Brüche mit steigendem q auch größer, was also zusätzlich Speicher belegt und Rechenzeit beansprucht. Nach den ersten Rechnungen wurde schnell klar, dass die vollständige Berechnung der Farey-Folgen zu ineffizient ist und deshalb die Berechnung ausschließlich des relevanten Teils der Farey-Folge, wie es in Beispiel 3.2.6 beschrieben ist, umgesetzt wurde. Dies brachte eine signifikante Effizienzsteigerung mit sich.

4.3 Auswertung der Daten

Alle nachfolgend aufgeführten Diagramme sind gleichermaßen als Diagramm mit Punktwerten dargestellt, die nicht miteinander verbunden wurden. Abgebildete Linien entstehen durch nahe beieinanderliegende Punkte im Diagramm

4.3.1 Laufzeiten

Die Algorithmen erreichten die nachfolgend aufgeführten Laufzeiten für die Datensätze $3 \leq n \leq 10.000$:

Algorithmus	h	min	sek	ms
Binär-Algorithmus		33	22	336
Greedy-Algorithmus		40	53	499
Farey-Folgen-Algorithmus	62	33	38	136

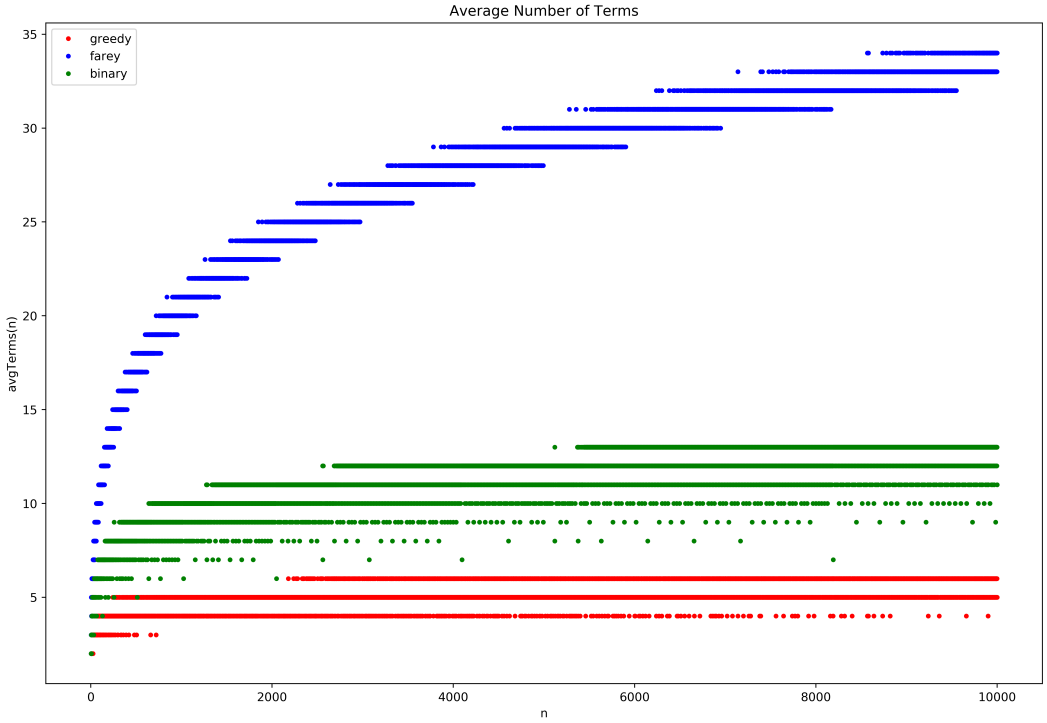
Tabelle 8: Laufzeitvergleich der Algorithmen

Die starke Unterlegenheit des Farey-Folgen-Algorithmus ist klar erkenntlich, allerdings nicht mehr wesentlich zu verbessern. Dieses Ergebnis entstand schon mit der in Beispiel 3.2.6 beschriebenen Methode, nur die relevante Farey-Folge zu konstruieren, da die Laufzeit mit vollständiger Konstruktion der jeweiligen Farey-Folge für jeden Datensatz n noch höher war. Es lässt sich folgern, dass er damit der laufzeittechnisch schlechteste der drei Algorithmen ist.

4.3.2 Durchschnittliche Anzahl der Terme

Eines der wichtigsten Ergebnisse zur realistischen Abschätzung der Algorithmen gegeneinander stellt die durchschnittliche Anzahl der Terme dar, die jeweils durch die Zerlegung entstehen. Abbildung 1 zeigt diese Entwicklung graphisch. Anhand der Graphen wird das logarithmische Wachstum der Werte erkenntlich. Dabei wird schnell deutlich, dass das Wachstum des Farey-Folgen-Algorithmus bedeutend größer ist als das der anderen beiden Algorithmen. Der Greedy- und der Binär-Algorithmus unterscheiden sich voneinander nicht so stark, offensichtlich schneidet in diesem Kriterium aber der Greedy-Algorithmus mit dem besten Ergebnis ab.

Abbildung 1: Durchschnittliche Anzahl der Terme in Abhängigkeit von n



4.3.3 Minimale Anzahl der Terme

Bezüglich der minimalen Anzahl der produzierten Terme liefert der Greedy-Algorithmus für alle n den gleichen Wert $minTerms_{greedy}(n) = 2$, was jeweils am Ergebnis des Bruchs $\frac{2}{n}$, liegt, welches immer 2 Summanden hat, da der Greedy-Algorithmus jeweils nach der Rechenvorschrift aus Abschnitt **??** vorgeht. Der Farey-Folgen-Algorithmus liefert ständig zwischen 2 und 3 alternierende Werte, wobei $n = 6$ eine Ausnahme liefert mit $minTerms_{farey}(6) = 5$, da aufgrund der vielen Kürzungen im Datensatz für $n = 6$ nur $\frac{5}{6}$ betrachtet wird, welcher in der Farey-Zerlegung 5 Terme benötigt. Beim Binär-Algorithmus erreichen die meisten Tests ein Minimum von 2 Termen pro Datensatz, in ca. $0,7\%$ der Fälle liegt das Minimum jedoch bei 3 Termen. Ein besonderer Zusammenhang zwischen solchen Datensätzen wurde nicht gefunden.

4.3.4 Maximale Anzahl der Terme

Der Greedy-Algorithmus pendelt sich ab $3.000 \leq n \leq 10.000$ bei Werten von $7 \leq maxTerms_{greedy}(n) \leq 16$ ein, während der Binäralgorithmus sich ab $2000 \leq n \leq 10.000$ mit Werten von $15 \leq maxTerms_{binary}(n) \leq$

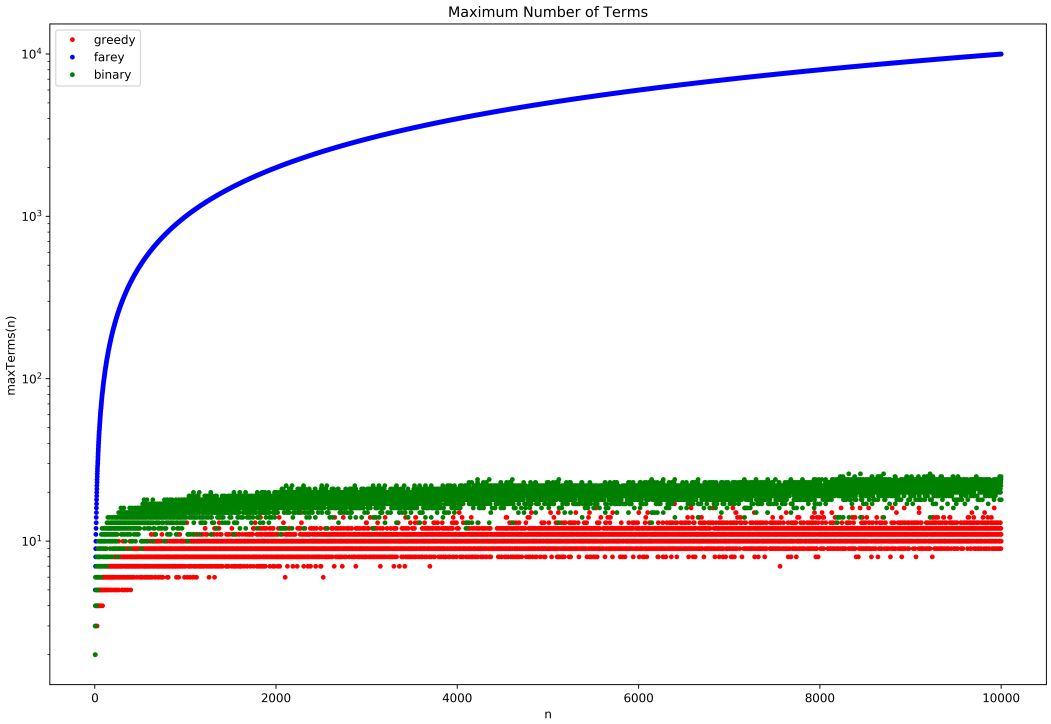
22 einpendelt, wobei einzelne Abweichungen nach oben und unten vernachlässigt wurden. Für den Farey-Folgen-Algorithmus hingegen gilt ein lineares Wachstum:

$$maxTerms_{farey}(n) = n + 1, \forall n$$

für alle getesteten Datensätze.

finde Erklär

Abbildung 2: Maximale Anzahl der Terme im Datensatz für jedes n

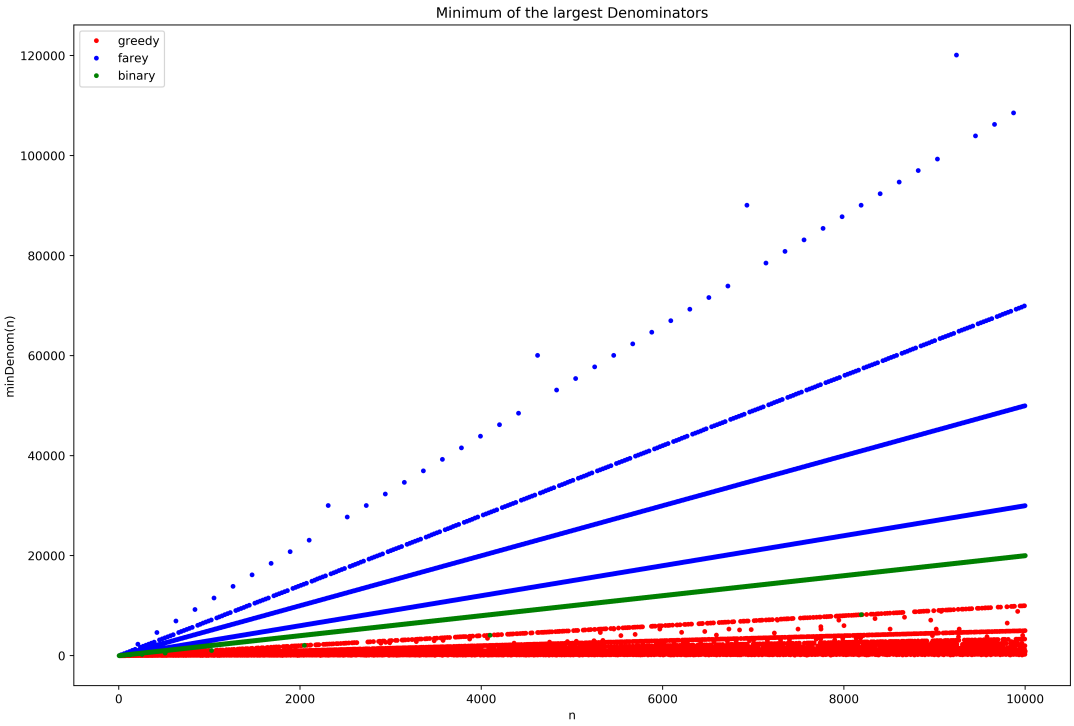


4.3.5 Minimum der größten Nenner

Bezeichne im Folgenden $u \in \mathbb{Q}$ den Anstieg der Geraden in der Funktion $f(n) = u \cdot n$.
Bezüglich des Minimums der größten Nenner zeigen alle Algorithmen klares lineares Wachstum, allerdings in sehr unterschiedlichen Ausprägungen.
Die Datenpunkte für den Binäralgorithmus ergeben zwei Geraden, eine für alle Werte, bei denen n Zweierpotenz ist, und eine für alle anderen Werte. Das liegt am Sonderfall 2 aus Algorithmus 3.3.1, der Brüche mit Nennern, die Zweierpotenzen sind, gesondert behandelt und dadurch kleinere Nenner in der Zerlegung produziert. Das lineare Wachstum der beiden Geraden entsteht durch das wachsende q in der Gleichung $\frac{p}{q} = \frac{s}{N_k} + \frac{r}{qN_k}$. Die Gerade der Zweierpotenzen wächst dabei mit Anstieg $u = 1$, die der nicht-Zweierpotenzen mit Anstieg $u = 2$.
Der Greedy-Algorithmus, dessen Werte bzgl. dieses Kriteriums unterhalb und zum Teil auf denen des

Binäralgorithmus liegen, zeigt hingegen mehr als zwei Geraden auf, also entwickeln sich bestimmte Folgen innerhalb der Testreihe dieses Algorithmus gleich, die Folgen sind aber in ihrer Entwicklung voneinander unterscheidbar. Die sich ergebenden Geraden haben einen Anstieg $u \in \left\{2, 1, \frac{1}{2}, \frac{1}{4}, ...\right\}$, einige Werte liegen aber auch zwischen diesen Geraden. Genaue Gründe hierfür konnten nicht gefunden werden.
Gleiches gilt für den größere Werte ergebenden Farey-Folgen-Algorithmus, für den sich 5 Geraden ergeben, allerdings wird hier die Anzahl der Datenpunkte, die jeweils eine Gerade ergeben, mit wachsendem Anstieg immer weniger. Der Anstieg der Geraden ist dabei immer eine Primzahl, die unterste Datenreihe liegt auf der Funktion $f(x) = 2x$, die oberste auf der Funktion $f(x) = 13x$. warum?

Abbildung 3: Minimum des größten Nenners im Datensatz für jedes n

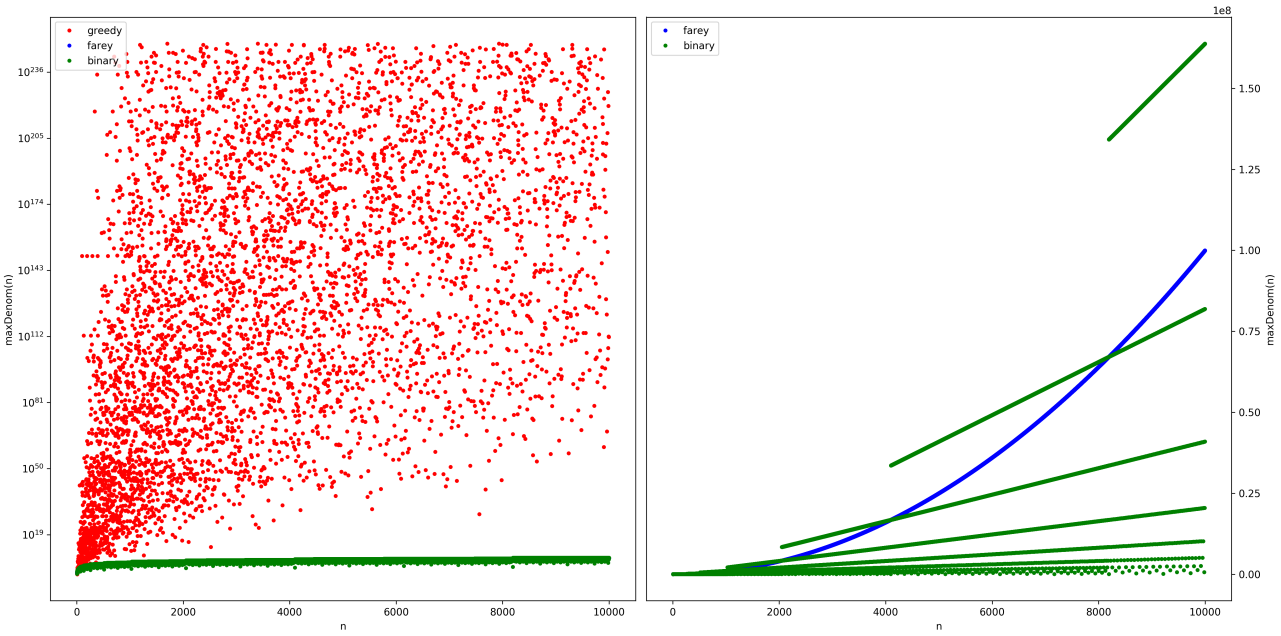


4.3.6 Maximum der größten Nenner

Dieses Kriterium ist jenes, in welchem sich die Algorithmen wohl am stärksten voneinander unterscheiden. Während sich die Werte von Binär- und Farey-Folgen-Algorithmus unter 10^{19} halten, streuen sich die Werte des Greedy-Algorithmus von $maxDenom_{greed}(3) = 2$ bis zu einem Maximum von $maxDenom_{greedy}(4967) \approx 7,3378 \times 10^{225.516}$ ohne jegliche erkennbare Regelmäßigkeit. In Abbildung 4 musste auf die Darstellung der 4991 größten Werte des Greedy-Algorithmus verzichtet werden, da diese Werte größer als 10^{250} und somit für die Verarbeitung mit *pyplot* zu groß waren.
Das Wachstumsverhalten des Binäralgorithmus scheint hier besonders interessant, da sich mehrere

Geraden ergeben, die zudem ein sprunghaftes Verhalten aufweisen. Das liegt daran, dass sich der größte Nenner im Binäralgorithmus aus dem Produkt qN_k ergibt, wobei das im Diagramm beschriebene $n = q$ ist. Das N_k ist immer die nächstgrößere Zweierpotenz von q, falls q nicht selbst schon eine solche ist. Daher springen die Werte immer bei q, falls q Zweierpotenz ist. Das stetig steigende q sorgt zudem für das lineare Wachstum der Geraden, den Anstieg bestimmt N_k.
Der Farey-Folgen-Algorithmus zeigt für dieses Kriterium ein quadratisches Wachstum auf. warum?

Abbildung 4: Maximum der größten Nenner pro Datensatz, links: mit Graph für den Greedy-Algorithmus, rechts: ohne selbigen



4.3.7 Zusammenhang innerhalb des Binäralgorithmus

Im Binary-Algorithmus konnte eine unerwartete Korrelation zwischen n und $minDenom(n)$ nachgewiesen werden:

$$minDenom_{binary}(n) = \begin{cases} n & \text{falls n Zweierpotenz ist} \\ 2n & \text{sonst.} \end{cases}$$

Begründen lässt sich das mit der Zerlegung des Zählers im Schritt 2 des Algorithmus 3.3.1, für den bei ungeraden Nennern in der Zerlegung immer eine 1 vorkommt, die nicht mit dem Zähler n gekürzt werden kann und somit den kleinsten Bruch $\frac{1}{n}$ bildet. Falls der Zähler des zu zerlegenden Bruchs gerade ist, kann durch das Kürzen der Nenner nur kleiner werden. Für alle n, die Zweierpotenz sind, folgt $minDenom_{binary}(n) = n$. Für den Fall, dass n keine Zweierpotenz ist, war kein direkter Zusammenhang erkenntlich, mittels welcher Brüche das Ergebnis $minDenom_{binary}(n) = 2n$ zustande kommt.

4.3.8 Vergleich mit Erwartungswerten

Zur Überprüfung der in Tabelle 7 aufgezeigten oberen Schranken sollen diese Werte mit den Ergebnissen der zuvor beschriebenen Testreihen verglichen werden.

Anzahl der Summanden Die oberen Schranken für sowohl Greedy- als auch Farey-Folgen-Algorithmus sind hier mit p, der Zähler des betrachteten Bruchs, angegeben, beim Binäralgorithmus ist hier nur die Komplexitätsklasse $O(\log q)$ angegeben. Bei Betrachtung der Ergebnisse, die in 2 zusammengefasst sind, zeigen sich der Farey-Folgen- und der Binäralgorithmus mit genau dem erwarteten Verhalten, nur der Greedy-Algorithmus bleibt weit unter seiner theoretischen Schranke und weist anstelle eines linearen Wachstums nur logarithmisches auf.

Größe der Nenner Die größtmöglichen Nenner für den Binäralgorithmus werden mit $2(q^2 - q)$, für den Farey-Folgen-Algorithmus mit $q(q - 1)$, angegeben. Wie an Abbildung 4 zu erkennen ist, halten sowohl Farey-Folgen- wie auch Binäralgorithmus ihre jeweiligen oberen Schranken ein. Der Greedy-Algorithmus liefert, wie Bleicher und Erdös in ihrem Artikel "Denominators of Egyptian Fractions"[3] schon berichteten, ein exponentielles Wachstum der Nenner, die aufgrund ihrer Größe in Abbildung 4 teilweise nicht dargestellt werden konnten.

4.4 Zusammenfassung

Der Greedy-Algorithmus ist der wohl schlechteste der Algorithmen, obwohl er bezüglich der Anzahl der Terme die besten Ergebnisse liefert. Grund dafür ist die Wahl des jeweils größten Nenners, wodurch es zu den extremen numerischen Ausbrüchen der Nenner kommt, die eine effiziente Nutzung des Algorithmus mittels normaler Computersysteme unmöglich macht, da die Standards vieler Programmiersprachen solch große bzw. kleine Zahlen nicht unterstützen. Durch die großen Schwankungen zwischen den Werten des Greedy-Algorithmus ist dieser zudem sehr unvorhersehbar. Obwohl er in den meisten Fällen durchaus brauchbare Ergebnisse liefert, ist nicht vorhersagbar, wann dies nicht der Fall ist. Im Vergleich dazu tendiert der Farey-Folgen-Algorithmus eher dazu, kleinere Nenner zu erzeugen, benötigt dafür im Durchschnitt aber deutlich mehr Terme. Der Binäralgorithmus bewegt sich in den verglichenen Kriterien meist zwischen den Werten der anderen Algorithmen, aber immer zum geringeren tendierend, von wenigen Punkten der maximalen Nenner abgesehen, in denen er dem Farey-Folgen-Algorithmus leicht unterlegen ist. Zudem hatte er im Vergleich die beste Laufzeit und nach allem Anschein auch den stabilsten Gesamtverlauf. Damit ist diese Berechnungsmethode wohl die gewinnbringendste, da sie mit relativ wenig Aufwand durchaus brauchbare Zerlegungen gemeiner in Ägyptische Brüche erzeugt.

[3] engl.: Nenner Ägyptischer Brüche, [Bleicher und Erdös, 1976, S. 157]

5 Theoretische Schranken für die maximale Anzahl der Stammbrüche

Nachdem untersucht wurde, was in der Praxis mit ausgewählten Methoden erreicht werden kann, soll nun das Licht auf einen weiteren sehr interessanten Aspekt Ägyptischer Brüche gelenkt werden, nämlich den Dingen, die theoretische herleitbare Schranken haben. Vor allem im 20. Jahrhundert wurde von zahlreichen Mathematikern an dem Problem, solche theoretischen Grenzen zu finden und zu beweisen, gearbeitet, unter ihnen Bleicher, Erdös, Graham und andere [Guy, 1981, S.87 ff].

Satz 5.0.1. *Sei $n \in \mathbb{N}$ ungerade. $\frac{2}{n}$ lässt sich für jedes n als Summe zweier Stammbrüche notieren, nämlich:*

$$\frac{2}{n} = \frac{1}{\lceil \frac{n}{2} \rceil} + \frac{1}{n \cdot \lceil \frac{n}{2} \rceil}.$$

Beweis. Sei $m \in \mathbb{N}$ so gewählt, dass $n = 2m + 1$. Es folgt:

$$\begin{aligned}
\frac{1}{\lceil \frac{n}{2} \rceil} + \frac{1}{n \cdot \lceil \frac{n}{2} \rceil} &= \frac{1}{m+1} + \frac{1}{(2m+1)(m+1)} \\
&= \frac{2m+1}{(2m+1)(m+1)} + \frac{1}{(2m+1)(m+1)} \\
&= \frac{2m+2}{(2m+1)(m+1)} \\
&= \frac{2(m+1)}{(2m+1)(m+1)} \\
&= \frac{2}{2m+1} \\
&= \frac{2}{n}.
\end{aligned}$$

□

Da der Term $\frac{1}{\lceil \frac{n}{2} \rceil}$ genau dem größten Stammbruch entspricht, der kleiner als $\frac{2}{n}$ ist, liefert der Greedy-Algorithmus hier für alle $\frac{2}{n}$ das gleiche Ergebnis wie die Rechenvorschrift aus Satz 5.0.1. Damit lässt sich auch sofort eine weiterer Satz aufstellen, diesmal für $\frac{3}{n}$.

Satz 5.0.2. *Für jedes beliebige $n \in \mathbb{N}$ lässt sich $\frac{3}{n}$ als Summe von höchstens 3 Stammbrüchen schreiben.*

Beweis. Es gilt

$$\frac{3}{n} = \frac{1}{n} + \frac{2}{n}.$$

Falls n gerade ist, reichen sogar nur zwei Stammbrüche, denn dann kann $\frac{2}{n} = \frac{1}{\frac{n}{2}}$ geschrieben werden; anderenfalls wird $\frac{2}{n}$ wie nach Satz 5.0.1 zerlegt und es ergeben sich genau drei Stammbrüche:

$$\frac{3}{n} = \frac{1}{n} + \frac{1}{\lceil \frac{n}{2} \rceil} + \frac{1}{n \cdot \lceil \frac{n}{2} \rceil}.$$

□

Bleicher und Erdös vermuteten, dass für alle $n \in \mathbb{N}$ der Bruch $\frac{4}{n}$ in höchstens 3 Stammbrüche zerlegt werden kann, was nicht formal bewiesen, aber von Nicola Franceschine zumindest für $n < 10^8$ gezeigt wurde. Die gleiche Vermutung für $\frac{5}{n}$ wurde durch W. Sierpiński aufgestellt, zunächst durch G. Palamá für alle $n < 922.321$ gezeigt und später durch Stewart auf alle $n < 1.057.438.801$, $n \not\equiv 1 \pmod{278.460}$ erweitert. Schinzel bewies zudem, dass der Ausdruck

$$\frac{4}{at+b} = \frac{1}{x(t)} + \frac{1}{y(t)} + \frac{1}{z(t)}$$

für Polynome $x(t), y(t)$ und $z(t)$ mit ganzzahligen Koeffizienten genau dann gilt, wenn b nicht quadratischer Rest modulo a ist. [Guy, 1981, S. 88]

Das Beispiel $\frac{4}{n}$, für welches noch immer nicht klar ist, ob es sich für alle $n \in \mathbb{N}$ durch 3 Stammbrüche als Ägyptischer Bruch darstellen lässt, zeigt auf, wie viel in diesem Bereich der Mathematik noch unbekannt ist. Das Problemfeld der Ägyptischen Brüche, das vor knapp 4000 Jahren aufgeworfen wurde, stellt auch die heutigen Mathematiker noch vor viele ungelöste Aufgaben und Fragen.

neu schreibe

6 Anhang

Zur Berechnung der Testreihen aus Abschnitt 4 wurde ausschließlich der nachfolgend aufgeführte, selbst entwickelte Code in PARI/GP The PARI Group [2018], Version *2.9.4* umgesetzt.

6.1 Hilfsfunktionen

Alle genutzten, nicht in PARI/GP enthaltenen Funktionen sind im Folgenden aufgelistet.

listsum(list) berechnet die Summe eines Verbunddatentyps, bspw. einer Liste oder eines Vektors. *listmin(list)* berechnet das Minimum einer Liste, *listmax(list)* das Maximum.

```
listsum( list ) = sum(i=1,#list,list[i]);
listmin( list ) = {local(minimum); minimum=list[1]; for(i=1,#list,if( list[i]<minimum, minimum = list[i])); return(
    minimum)}
listmax( list ) = {local(maximum); maximum=list[1]; for(i=1,#list,if(list[i]>maximum, maximum = list[i])); return(
    maximum)}
```

contains(list, element) durchsucht einen Verbunddatentyp und gibt 1 zurück, falls *element* in *list* enthalten ist, 0 sonst.

```
contains( list , element)={
  for(i=1, #list,
    if(element == list[i],
      return(1);
    );
  );
  return(0);
}
```

reversevecsort(vect) sortiert *vect* in absteigender Reihenfolge.

```
reverse_vecsort(vect)={
  local(result);
  result = List();
  vect = vecsort(vect);
  forstep(i = #vect, 1, -1, listput(result, vect[i]));
  return(Vec(result));
}
```

FareySeries(order) berechnet die Farey-Folge der Ordnung *order* und gibt diese als Vektor zurück.

```
FareySeries(order)={
  local(result);
  result = List();
  for(den=1,order,
    for(num=0, den,
      listput(result, num/den);
```

```
    );
  );
  result = vecsort(Vec(Set(result)));
  return(result);
}
```

findAdjacent(Fs, fraction) sucht in der Farey-Folge *Fs* den adjazenten Bruch zu *fraction*, der kleiner ist. Diese Funktion wird seit der Implementierung von *findAdjacentFrel* nicht mehr verwendet und dient seither nur dem Vergleich.

```
findAdjacent(Fs, fraction)={
  /*lb: lower bound, ub: upper bound, index: index of currently examiined fraction in Fq*/
  local(lb,ub,index);
  if(fraction == Fs[#Fs],
    return(Fs[#Fs-1]);
  );
  lb=1;
  ub=#Fs;
  index = #Fs\2;
  while(Fs[index] != fraction,
    if(Fs[index] > fraction,
      ub=index;
      index = (ub-lb)\2+lb,
    /*else*/
      lb = index;
      index = (ub-lb)\2+lb;
    );
  );
  return(Fs[index-1]);
}
```

findAdjacentFrel(fract) berechnet den kleineren der zu *fract* adjazenten Brüche mithilfe des relevanten Teils der Farey-Fogle, wie in Beispiel 3.2.6 gezeigt und stellt damit die wesentlich effizientere Alternative zu *findAdjacent(Fs,fraction)* dar.

```
adjacentFrel(fract)={
  /*lb: lower bound, ub: upper bound, med: mediant fraction of ub and lb*/
  local(lb,ub,med);
  if(fract <= 0/1 || fract >= 1/1, return(0/1););
  lb=0/1;
  ub=1/1;
  med = mediant(lb,ub);
  while(fract != med,
    if(fract < med,
      ub = med,
      lb = med;
    );
    med = mediant(lb,ub);
  );
```

```
15   return(lb);
```

mediant(frac1, frac2) berechnet die Mediante zweier Brüche nach Definition 3.2.5.

```
1 mediant(frac1, frac2)={return((numerator(frac1)+numerator(frac2))/(denominator(frac1)+denominator(frac2)));}
```

findDivisorsOf_k_addingup_n(k,n) sucht jene Teiler von k heraus, die in ihrer Summe n ergeben.

```
1  findDivisorsOf_k_addingup_n(k,n)={
2    local(result);
3    if(k<n,return(Vec([-1])));
4    res = List();
5    while((listsum(res) != n) && (k >= 1),
6      if(listsum(res) + k <= n,
7        listput(res,k);
8      );
9      k /= 2;
10   );
11   return(res);
12 }
```

printEgypFrac(arguments) nimmt eine Liste mit Argumenten entgegen und gibt die String-Konkatenation dieser zurück.

```
1  printEgypFrac(arguments)={
2    local(result);
3    result = "";
4    if(#arguments <= 0,
5      result = "0";
6      return(result);
7    );
8    if(#arguments == 1,
9      result = Str(arguments[1]);
10     return(result);
11   );
12   for(i=1, #arguments-1,
13     result = Str(result, Str(arguments[i]));
14     result = Str(result, " + ");
15   );
16   result = Str(result, arguments[#arguments]);
17   return(result);
18 }
```

6.2 PARI/GP Code für den Greedy-Algorithmus

Die Funktion *fibonacci_sylvester(fraction, stepsize, start)* berechnet mittels des gleichnamigen Algorithmus die entsprechende Ägyptische Darstellung des Bruchs *fraction*. Das Argument *stepsize* gibt an,

um wie viel der Nenner eines Kandidaten bei Bedarf mindestens erhöht wird. Der zuerst untersuchte Kandidat ist $\frac{1}{\text{start}}$. Damit wird die Funktionalität zur Verfügung gestellt, bspw. nur nach Stammbrüchen mit geraden oder ungeraden Nennern zu suchen.
Letztendlich ruft der Nutzer aber nur die Stellvertreterfunktionen *greedy*, *greedy_odd* oder *greedy_even* auf, die die Hauptfunktion *fibonacci_sylvester* mit Standardwerten für *stepsize* und *start* nutzen.

```
fibonacci_sylvester(fraction, stepsize, start)={
  /*candidate: current candidate, result: list with resulting unit fractions*/
  local(candidate, result);
  result=List();
  candidate = start;

  /* print error if fraction is larger one */
  if (numerator(fraction) > denominator(fraction),
    print("fraction is larger than 1. Use fractions smaller or equal to 1.");
    return;
  );

  /* check if fraction is a unit fraction */
  if (numerator(fraction) == 1,
    print("fraction was already a unit fraction");
    listput(result, fraction);
  );

  /* calculate summands and add them to the result */
  while (listsum(result) < fraction,
    candidate += stepsize;
    while (1/candidate > fraction - listsum(result),
      candidate += stepsize;
    );
    print("adding ", 1/candidate);
    listput(result, 1/candidate);
  );
  print("Greedy: ", fraction, " = ", printEgypFrac(result));
  return(Vec(result));
}

greedy(fraction) = fibonacci_sylvester(fraction, 1, 1);
greedy_odd(fraction) = {print("\nthis might not come to an end!\n");
      alarm(3600, fibonacci_sylvester(fraction, 2, 1));}
greedy_even(fraction) = fibonacci_sylvester(fraction, 2, 0);
```

6.2.1 Optimierter Greedy-Algorithmus

Zur Effizienzsteigerung wurde der Greedy-Algorithmus nochmals mit einer wesentlich effizienteren Suche der Nenner umgesetzt, die sich mit dem Namen ”Double & Add” beschreiben lässt und dem Prinzip der Ägyptischen Multiplikation aus Abschnitt 2.1 entspricht. Die Funktionalität der Suche des

größten Stammbruchs $max\{\frac{1}{n} : n \in \mathbb{N}\}$, der kleiner als ein gegebener rationaler Bruch ist, wurde in die Funktion *largestUnitFractionLEQ* ausgelagert. Der Rest des Algorithmus läuft ab wie im Anhang 6.2 beschrieben.

```
largestUnitFractionLEQ(fraction)={
  /* x: current candidate, ub: currently known upper bound*/
  local(x,ub);
  x=2;
  ub=0;

  /*catch unit fractions for efficiency */
  if(numerator(fraction) == 1,
    return(fraction);
  );

  while(1/x > fraction,
    x*=2;
  );

  ub=x;

  while(x > 1,
    x /= 2;
    if(1/(ub-x) <= fraction,
      ub -= x;
    );
  );
  return(1/ub);
}

greedy_fast(fraction)={
  local(result);
  result = List();
  /* print error if fraction is larger one */
  if (numerator(fraction) > denominator(fraction),
    print("fraction is larger than 1. Use fractions smaller or equal to 1.");
    return;
  );

  /* check if fraction is a unit fraction */
  if (numerator(fraction) == 1,
    print("fraction was already a unit fraction");
    listput(result, fraction);
  );

  /* calculate summands and add them to the result */
  while (listsum(result) < fraction,
    listput(result, largestUnitFractionLEQ(fraction-listsum(result)));
  );
  print("Greedy fast:\t", fraction, " = ", printEgypFrac(result));
  return(Vec(result));
```

```
48 }
```

6.3 PARI/GP Code für den Farey-Folgen-Algorithmus

Die Implementierung des Farey-Folgen-Algorithmus greift auf eine Dauerschleife zurück, aus der ausgebrochen wird, sobald der Nenner des betrachteten adjazenten Bruchs 1 ist, was laut Algorithmus 3.2.3 das Abbruchkriterium ist. *adjacent* ist dem Namen entsprechend der aktuelle, kleinere adjazente Bruch zum aktuell untersuchten Bruch *current_fraction*, *remainder* ist $\frac{1}{qs}$ aus demselben Algorithmus. Wie üblich stellt *result* die Liste der Ergebnissummanden dar.

```
1  FS(fraction)={
2    /*adjacent: the adjacent fraction to current_fraction, remainder: 1/qs */
3    local(adjacent, remainder, result, current_fraction);
4    result = List();
5    current_fraction = fraction;
6    while(1,
7      /*old: adjacent = findAdjacent(FareySeries(denominator(current_fraction)), current_fraction);*/
8      adjacent = adjacentFrel(current_fraction);
9      remainder = 1/(denominator(current_fraction)*denominator(adjacent));
10     listput(result, remainder);
11     if(numerator(adjacent) == 1,
12       listput(result, adjacent);
13       result = reverse_vecsort(Vec(result));
14       /*print("Farey_Sequence:\t", fraction, " = ", printEgypFrac(result));*/
15       return(result);
16     );
17     current_fraction = adjacent;
18   );
19   return(-1);
20 }
```

6.4 PARI/GP Code für den Binäralgorithmus

p und q entsprechen Zähler und Nenner des Bruchs *fraction*; r und s sind die natürlichen Zahlen, aus denen sich gemäß Algorithmus 3.3.1 $qs+r = pN_k$ ergibt. Die Liste *summands* enthält die Summanden der Zweierpotenzen, die in Summe den Zähler des aktuell betrachteten Bruchs ergeben. Das Ergebnis ist die Liste der Summanden des Ägyptischen Bruchs, die in *result* gespeichert wird.

```
1  binary_algo(fraction)={
2    /*summands: the divisors of Nk which add up to p or s, r respectively*/
3    local(p,q,r,s,Nk,summands,result);
4    result = List();
5    p = numerator(fraction);
6    q = denominator(fraction);
7    Nk=1;
8    while(q > Nk, Nk*=2;);
```

```
9   if(q == Nk,
10    summands = findDivisorsOf_k_addingup_n(Nk,p);
11    for(i=1, #summands, listput(result, 1/(Nk/summands[i]))),
12  /*else*/
13    [s,r] = divrem(p*Nk,q);
14    summands = findDivisorsOf_k_addingup_n(Nk,s);
15    for(i=1, #summands, listput(result, 1/(Nk/summands[i])));
16    summands = findDivisorsOf_k_addingup_n(Nk,r);
17    for(i=1, #summands, listput(result, 1/((q*Nk)/summands[i])));
18  );
19  /*print("Binary:\t\t", fraction, " = ", printEgypFrac(result));*/
20  return(reverse_vecsort(Vec(result)));
21 }
```

Literaturverzeichnis

[Beck 2000] Beck, Anatole: *Excursions Into Mathematics - The Millennium Edition*. Wellesley, Massachusetts : Peters, 2000. – ISBN 978–1–568–81115–4

[Bleicher 1972] Bleicher, M. N.: A new algorithm for the expansion of Egyptian fractions. In: *Journal of Number Theory, vol. 4, no. 4, pp. 342-382* 4 (1972), August, S. 342–382

[Bleicher und Erdös 1976] Bleicher, M. N. ; Erdös, P.: Denominators of Egyptian Fractions. In: *Journal of Number Theory, vol. 8, no. 8, pp. 157-168* 8 (1976), Mai, S. 157–168

[Burton 2011] Burton, David: *The History of Mathematics - An Introduction*. 7. Auflage. New York : McGraw-Hill, 2011. – 33–46 S. – ISBN 978–0–071–28920–7

[Gong 1992] Gong, Kevin: Egyptian Fractions. In: *Math 196 Spring, UC Berkeley* (1992)

[Guy 1981] Guy, Richard K.: *Unsolved Problems in Number Theory*. New York : Springer-Verlag, 1981. – 87–93 S.

[Resnikoff und Wells 1984] Resnikoff, H. L. ; Wells, R. O.: *Mathematics in Civilization* -. Subsequent. New York : Dover Publications, 1984. – ISBN 978–0–486–24674–1

[The PARI Group 2018] The PARI Group: *PARI/GP, version 2.9.4*. Univ. Bordeaux, 2018. – available from `http://pari.math.u-bordeaux.fr/`

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.
Ferner habe ich vom Merkblatt über die Verwendung von Bachelor/Masterabschlussarbeiten Kenntnis genommen und räume das einfache Nutzungsrecht an meiner Bachelorarbeit der Universität der Bundeswehr München ein.

Neubiberg, den 19.11.2019

Lars Berger