

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

## Inhaltsverzeichnis














<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Ägyptische Arithmetik</b>	<b>3</b>
2.1	Ägyptische Multiplikation . . . . .	3
2.2	Ägyptische Division . . . . .	3
2.2.1	Ganzzahlige Division . . . . .	3
2.2.2	Division mit Rest . . . . .	4
2.3	Ermittlung Ägyptischer Zerlegungen von Brüchen . . . . .	5
<b>3</b>	<b>Algorithmen zur Erstellung Ägyptischer Brüche</b>	<b>7</b>
3.1	Der Greedy-Algorithmus . . . . .	7
3.2	Der Farey-Folgen-Algorithmus . . . . .	8
3.3	Binäralgorithmus . . . . .	10
3.4	Beispielrechnungen . . . . .	12
3.5	Fazit und Vergleich . . . . .	14
<b>4</b>	<b>Auswertung einiger Testreihen</b>	<b>16</b>
4.1	Methodik . . . . .	16
4.2	Anfängliche Probleme und Effizienzsteigerung während der Implementierung . . . . .	16
4.3	Auswertung der Daten . . . . .	17
4.3.1	Laufzeiten . . . . .	17
4.3.2	Durchschnittliche Anzahl der Terme . . . . .	17
4.3.3	Minimale Anzahl der Terme . . . . .	18
4.3.4	Maximale Anzahl der Terme . . . . .	18
4.3.5	Minimum der größten Nenner . . . . .	18
4.3.6	Maximum der größten Nenner . . . . .	19
4.3.7	Zusammenhang innerhalb des Binary-Algorithmus . . . . .	19
4.4	Zusammenfassung . . . . .	19
<b>5</b>	<b>Theoretische Grenzen</b>	<b>20</b>
5.1	Theoretische Grenzen für $\frac{2}{n}$ . . . . .	20
<b>6</b>	<b>Anhang</b>	<b>21</b>
6.1	Hilfsfunktionen . . . . .	21
6.2	PARI/GP Code für den Greedy-Algorithmus . . . . .	23
6.2.1	Optimierter Greedy-Algorithmus . . . . .	24
6.3	PARI/GP Code für den Farey-Folgen-Algorithmus . . . . .	26
6.4	PARI/GP Code für den Binäralgorithmus . . . . .	26
	<b>Literaturverzeichnis</b>	<b>28</b>

**Eidesstattliche Erklärung**

**29**

## TODOs

## Notes

 insert ref . . . . .	2
 Tabellen 1 und 2 nebeneinander anordnen(?) . . . . .	4
 vielleicht doch? . . . . .	5
 eigenes Kapitel/ Subkapitel für "Rechentricks" der Ägypter? . . . . .	6
 Quelle . . . . .	7
 Beispielrechnung . . . . .	10
 Komplexitätsabschätzung! . . . . .	11
 insert graphics . . . . .	18
 insert graphics . . . . .	18
 finde Erklärung . . . . .	18
 insert graphics . . . . .	18
 insert graphics . . . . .	18
 insert graphics . . . . .	19

## 1 Einleitung

Vor über 4000 Jahren entstand in Ägypten das heute als "Rhind-Papyrus" bekannte Dokument, das als älteste bekannte Schrift mathematischen Wissens der Menschheit gilt. In der Präambel dieses Papyrus heißt es "Ein sorgfältiges Studium aller Dinge, Einblick in Alles, was es gibt, Wissen über alle obskuren Geheimnisse" <sup>1</sup>.

In den dort enthaltenen 85 Problemen werden dann Multiplikation und Division definiert sowie darauf aufbauende Probleme diskutiert. Obwohl heute bekannt ist, dass die Arithmetik der Ägypter sich ab einem bestimmten Zeitpunkt nicht weiterentwickelte bzw. weiterentwickeln konnte, da sie kein Stellenwertsystem besaßen, bietet die aus heutiger Sicht primitive Mathematik des alten Ägypten viele bis heute ungelöste Probleme. Ein solches Problemfeld sind die sogenannten Ägyptischen Brüche.

...

**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.

Obwohl die Divergenz der harmonischen Reihe zeigt, dass man mit Brüchen solcher Art durchaus alle rationalen Zahlen  $x \in \mathbb{Q}$  erzeugen kann, waren für ganzzahlige Werte in Ägypten Schreibweisen gängig, 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.

<sup>1</sup> [BURTON, 2011, S. 37, Übersetzung durch den Autor]

## 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 ( $\checkmark$ ). Schließlich werden die Zahlen der rechten Spalte aufaddiert, deren Zeile mittels  $\checkmark$  markiert wurde. Die sich ergebende Summe ist das Ergebnis.

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

	$\checkmark$	1	23		$\checkmark$	1	23
						2	46
$\checkmark$		1	69		$\checkmark$	4	92
$\checkmark$		2	138			8	184
$\checkmark$		4	276			16	368
		8	552			32	736
$\checkmark$		16	1104		$\checkmark$	64	1472
Summe:		23	<u>1587</u>	Summe:		69	<u>1587</u>

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  $b \cdot x = a$ . Somit ergibt sich das Problem der Multiplikation, nur dass die unbekannte Variable eine andere ist.

**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
<hr/>		
	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 nötigen 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
<hr/>		
×	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:

Tabellen 1 und 2 nebeneinander anordnen(?)

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

Tabelle 1: Teilen durch 7, dann Fortgesetzte Halbierung von 1

Zudem ist auch das bloße fortgesetzte Halbieren der Zahl oben rechts praktisch angewendet worden:

1	7
$\frac{1}{2}$	$3 + \frac{1}{2}$
$\frac{1}{4}$	$1 + \frac{1}{2} + \frac{1}{4}$
$\frac{1}{8}$	$\frac{1}{2} + \frac{1}{4} + \frac{1}{8}$
$\vdots$	$\vdots$

Tabelle 2: Fortgesetzte Halbierung von 7

Verfahren wird hier nach einem systematisierten trial-and-error-Verfahren. Da sich aus Tabelle 1 die Harmonische Reihe ergibt, aber ein Gesamtwert von  $117 - 112 = 5$  benötigt wird, fällt die Wahl zunächst auf das größte Element in Tabelle 2 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 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
✓	$\frac{1}{14}$	$\frac{1}{2}$
<hr/>		
	$16 + \frac{1}{2} + \frac{1}{7} + \frac{1}{14}$	117

Tabelle 3: 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 einfach zu berechnen, waren  $\frac{1}{2^x}$ , sowie  $\frac{1}{n \cdot 2^x}$ .

Aus dieser Methodik heraus ergibt sich die Notation der Ägyptischen Brüche. Selbstverständlich wurde auch mit solchen Brüchen multipliziert und dividiert, solche Beispiele würden aber den Rahmen dieser Arbeit sprengen, weshalb diese nicht weiter betrachtet werden.

vielleicht  
doch?

## 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

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

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]



2 Ägyptische Arithmetik

---

Für z.B. Brüche  $\frac{2}{n}$  für  $5 \leq n \leq 101$  ungerade 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]

eigenes Kapitel/ Subkapitel für "Rechentricks" der Ägypter?

### 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:

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

#### 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. Dabei werden solange jeweils die größtmöglichen Stammbrüche  $\frac{1}{x_i}$  gesucht, sodass

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

wobei gilt, dass

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

bis

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

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  $\frac{1}{x} \leq \frac{p}{q}$ .

### 3 Algorithmen zur Erstellung Ägyptischer Brüche

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):

$$\frac{1}{x} \leq \frac{p}{q} < \frac{1}{x-1}. \quad (2)$$

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):

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

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

## 3.2 Der Farey-Folgen-Algorithmus

Eine weitere Methode zur Erstellung Ägyptischer Brüche stellt der sogenannte Farey-Folgen-Algorithmus dar, der seinen Namen aus dem Umstand bezieht, dass die Farey-Folge dafür genutzt wird.

3 Algorithmen zur Erstellung Ägyptischer Brüche

**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.  $\square$

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 Medianten zweier rationaler Brüche  $\frac{a}{b}, \frac{c}{d} \in \mathbb{Q}_+$   $medant : \mathbb{Q}_+ \times \mathbb{Q}_+ \rightarrow \mathbb{Q}_+$  sei definiert als:

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

**Beispiel 3.2.6.** Sei  $\frac{p}{q} = \frac{21}{23}$ . Die obere und untere Schranken sind in unserem Fall 0 bzw. 1. Die Medianten liegen 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 Medianten liegen nun bei  $\frac{2}{3}$ ,  $\frac{2}{3} < \frac{21}{23} < 1$  usw. Bei der Medianten  $\frac{11}{12}$  stellen wir fest:  $\frac{10}{11} < \frac{21}{23} < \frac{11}{12}$ . Daraus folgt der relevante Teil von  $F_{23}$ , hier  $F_{23rel}$  genannt:

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

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  $F_{23}$ .

Beispielrechnung

### 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  $s, r \in \mathbb{N}$ ,  $0 < r < N_k$  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.

3 Algorithmen zur Erstellung Ägyptischer Brüche

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  $\log_2 N = k$  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 zusammengekommen paarweise verschieden sind, folgt daraus, dass die zu  $s$  gehörenden Nenner immer Zweierpotenzen sind, die zu  $r$  gehörigen aber niemals.  $\square$

Sei  $\frac{p}{q} \in \mathbb{Q}_+$ ,  $\frac{p}{q} < 1$ . Gong zeigte, dass

Komplexitätsabschätzung!

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:

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

### 3 Algorithmen zur Erstellung Ägyptischer Brüche

Die Ergebnisse sind nochmals in Tabelle 4 zusammengefasst.

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.

**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  $F_3$ 1 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 5 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<sup>2</sup> 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.



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

**Der Greedy-Algorithmus** liefert

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

**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 6 zeigt, ist in diesem Beispiel der Greedy-Algorithmus leicht überlegen. An den Zweierpotenzen der ersten Temre des Binäralgorithmus lässt sich sehr gut dessen Natur erkennen, die ihm aber in diesem Beispiel nicht zu sonderlicher Effizienz verhilft.

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]

---

<sup>2</sup> greedy, engl. für: habgierig, gierig, gefräßig

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

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

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

Algorithmus	Anzahl der Summanden	Länge des längsten Nenners
Fibonacci-Sylvester(Greedy)	$p$	exponentiell
Farey-Folgen-Algorithmus	$p$	$q(q-1)$
Binäralgorithmus	$\log q$	$q^2$

Tabelle 7: Vergleich der beschriebenen Algorithmen (obere Schranken)

## 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 gerechnet.

### 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  $M_n = \{\frac{x}{n} \mid 1 \leq x < n \wedge x \nmid n\}$  und enthält neben dem  $n$  selbst folgende Eigenschaften, die zugleich den nachgestellten Namen bekommen:

- 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,  $maxDenom(n)$ .

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 der Funktion *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  $(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \dots, \frac{1}{x})$  gesucht, was für große Nenner offensichtlich sehr lange dauert. Nachdem eine Optimierung vorgenommen wurde, in der nun zunächst die Nenner der genannten Folge verdoppelt und dann beim Halbieren aufsummiert werden  $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots, \frac{1}{n_i}, \frac{1}{n_i+n_j}, \frac{1}{n_i+n_j+n_k}, \dots, \frac{1}{x})$ , wobei alle  $n$  Zweierpotenzen sind und  $n_i > n_j > n_k > \dots \geq$

#### 4 Auswertung einiger Testreihen

1;  $i, j, k \in \mathbb{N}$ , 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 diese Elemente mit steigender Ordnung auch größer, was also zusätzlich Speicher belegt und Rechenzeit beansprucht. Nach den ersten paar Rechnungen wurde schnell klar, dass die vollständige Berechnung der Farey-Folgen nicht ansatzweise optimal ist und deshalb der Ansatz der 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

#### 4.3.1 Laufzeiten

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

<i>Algorithmus</i>	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 zeigt graphisch die Entwicklung. Es ist klar zu erkennen, wie sich Greedy- und Binary-Algorithmus früh bei einem relativ festen Wert mit konstanten Schwankungen einpendeln, der Ausschlag der Kurve für den Farey-Folgen-Algorithmus aber signifikant stärker ist und sich erst später bei einem wesentlich höheren

#### 4 Auswertung einiger Testreihen

Wert einpendelt. Alle Algorithmen haben aber starke Schwankungen, es gibt also keinen festen Wert, der in bestimmten Bereichen eingenommen wird.

insert graphics

#### 4.3.3 Minimale Anzahl der Terme

Bezüglich der minimalen Anzahl der produzierten Terme liefert der Greedy-Algorithmus für alle  $n$  den gleichen Wert  $\minTerms_{greedy}(n) = 2$ . 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% Fälle liegt das Minimum jedoch bei 3 Termen. Ein besonderer Zusammenhang zwischen solchen Datensätzen wurde nicht gefunden.

insert graphics

#### 4.3.4 Maximale Anzahl der Terme

Der Greedy-Algorithmus pendelt sich ab  $n > 3000$  bei Werten von  $7 \leq \maxTerms_{greedy}(n) \leq 16$  ein, während der Binäralgorithmus sich ab  $n > 2000$  mit Werten von  $15 \leq \maxDenom_{binary}(n) \leq 22$  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 Erklärung

insert graphics

#### 4.3.5 Minimum der größten Nenner

In dieser Hinsicht zeichnet sich ein besonders interessantes Bild ab. Alle Algorithmen zeigen klares lineares Wachstum bezüglich  $\minDenom(n)$ , allerdings in sehr unterschiedlichen Ausprägungen. Der Binäralgorithmus bewegt sich zwischen den Werten der anderen Algorithmen und ist, bis auf wenige Ausnahmen, durch eine einzige Gerade erkenntlich. Der Greedy-Algorithmus, dessen Werte dieses Kriteriums unterhalb derer des Binäralgorithmus liegen, zeigt hingegen mehrere Geraden auf, also entwickeln sich bestimmte Folgen innerhalb der Testreihe dieses Algorithmus gleich, die Folgen sind aber in ihrer Entwicklung voneinander unterscheidbar. Gleiches gilt für den mit größeren Werten 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 selbiger, immer weniger. *Es ist zu vermuten, dass dies von gewissen Moduln abhängt, weitere Erkenntnisse fehlen noch*

insert graphics

#### 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  $\maxDenom_{greedy}(3) = 2$  bis zu einem Maximum von  $\maxDenom_{greedy}(4967) \approx 7,3378 \times 10^{225.516}$  ohne jegliche erkennbare Regelmäßigkeit. *binary & farey relativ regelmäßig (Vgl. quadratisch o.ä.?)*

insert graphics

#### 4.3.7 Zusammenhang innerhalb des Binary-Algorithmus

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

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

#### 4.4 Zusammenfassung

- greedy zwar der beste bzgl. #Terms, aber mit  $\maxDenom$  fast unbrauchbar  $\rightarrow$  sehr unvorhersehbar/unkontrollierbar
- Farey liefert zwar brauchbare Größenordnung für die Länge der Nenner, aber dafür extrem viele Terme
- Binär ist wohl der am stabilsten laufende und brauchbarste, zudem der schnellste der getesteten Verfahren.

## 5 Theoretische Grenzen

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 Grenzen haben. *Bemühungen Vieler, Grenzen für bestimmte Fälle zu finden => Quellen auslutschen.*

### 5.1 Theoretische Grenzen für $\frac{2}{n}$

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

## 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.

```
1 listsum( list ) = sum(i=1,#list,list[i]);
2 listmin( list ) = {local(minimum); minimum=list[1]; for(i=1,#list,if(list[i]<minimum, minimum = list[i])); return(
    minimum)}
3 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.

```
1 for(i=1, #list,
2     if(element == list[i],
3         return(1);
4     );
5 );
6 return(0);
7 }
```

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

```
1 local( result );
2 result = List();
3 vect = vecsort(vect);
4 forstep(i = #vect, 1, -1, listput( result , vect[i]));
5 return(Vec(result));
6 }
```

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

```
1 local( adjacent, remainder, result , current_fraction);
2 result = List();
3 current_fraction = fraction;
4 if(numerator(fraction) == 1, return(fraction));
5 while(1,
6 /* old: adjacent = findAdjacent(FareySeries(denominator(current_fraction)), current_fraction); */
7     adjacent = adjacentFrel(current_fraction);
8     remainder = 1/(denominator(current_fraction)*denominator(adjacent));
```



6 Anhang

```

9  listput (result , remainder);
10 if (numerator(adjacent) == 1,
11     listput (result , adjacent);
12     result = reverse_vecsort(Vec(result));
13     print("Farey_Sequence: ", fraction, " = ", printEgypFrac(result));
14     return(result);
15 );
16 current_fraction = adjacent;
17 );
18 return(-1);
19 }

```

*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.

```

1  local (lb,ub,index);
2  if (fraction == Fs[#Fs],
3      return(Fs[#Fs-1]);
4  );
5  lb=1;
6  ub=#Fs;
7  index = #Fs\2;
8  while(Fs[index] != fraction,
9      if (Fs[index] > fraction,
10         ub=index;
11         index = (ub-lb)\2+lb,
12         /*else*/
13         lb = index;
14         index = (ub-lb)\2+lb;
15     );
16 );
17 return(Fs[index-1]);
18 }

```

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

```

1  local (lb,ub,med);
2  if (fract <= 0/1 || fract >= 1/1, return(0/1));
3  lb=0/1;
4  ub=1/1;
5  med = mediant(lb,ub);
6  while(fract != med,
7      if (fract < med,
8         ub = med,
9         lb = med;
10     );

```

## 6 Anhang

```

11     med = median(lb,ub);
12 );
13 return(lb);
14 }

```

*median(frac1, frac2)* berechnet die Mediane zweier Brüche nach Definition 3.2.5.

*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, die die Funktion dann als Summe aller Elemente als String sowohl ausgibt als auch für eventuelle weitere Verarbeitung zurückgibt.

```

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

## 6 Anhang

an, um wie viel der Nenner eines Kandidaten bei Bedarf mindestens erhöht wird.  $\frac{1}{\text{start}}$  gibt den zuerst untersuchten Kandidaten an. 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.

```

1 fibonacci_sylvester(fraction, stepsize, start)={
2   local(candidate, result);
3   result=List();
4   candidate = start;
5
6   /* print error if fraction is larger one */
7   if (numerator(fraction) > denominator(fraction),
8     print("fraction is larger than 1. Use fractions smaller or equal to 1.");
9     return;
10  );
11
12  /* check if fraction is a unit fraction */
13  if (numerator(fraction) == 1,
14    print("fraction was already a unit fraction");
15    listput(result, fraction);
16  );
17
18  /* calculate summands and add them to the result */
19  while (listsum(result) < fraction,
20    candidate += stepsize;
21    while (1/candidate > fraction - listsum(result),
22      candidate += stepsize;
23    );
24    listput(result, 1/candidate);
25  );
26  print("Greedy: ", fraction, " = ", printEgypFrac(result));
27  return(Vec(result));
28 }
29
30 greedy(fraction) = fibonacci_sylvester(fraction, 1, 1);
31 greedy_odd(fraction) = {print("\nthis might not come to an end!\n");
32   alarm(3600, fibonacci_sylvester(fraction, 2, 1));}
33 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

## 6 Anhang

die Funktion *largestUnitFractionLEQ* ausgelagert. Der Rest des Algorithmus läuft ab wie im Anhang 6.2 beschrieben.

```

1  /* x: current candidate, ub: currently known upper bound*/
2  local(x,ub);
3  x=2;
4  ub=0;
5
6  /*catch unit fractions for efficiency */
7  if (numerator(fraction) == 1,
8      return(fraction);
9  );
10
11 while(1/x > fraction,
12     x*=2;
13 );
14
15 ub=x;
16
17 while(x > 1,
18     x /= 2;
19     if(1/(ub-x) <= fraction,
20         ub -= x;
21     );
22 );
23 return(1/ub);
24 }
25
26 greedy_fast(fraction)={
27     local(result);
28     result = List();
29     /* print error if fraction is larger one */
30     if (numerator(fraction) > denominator(fraction),
31         print("fraction is larger than 1. Use fractions smaller or equal to 1.");
32         return;
33     );
34
35     /* check if fraction is a unit fraction */
36     if (numerator(fraction) == 1,
37         print("fraction was already a unit fraction");
38         listput(result, fraction);
39     );
40
41     /* calculate summands and add them to the result */
42     while (listsum(result) < fraction,
43         listput(result, largestUnitFractionLEQ(fraction-listsum(result)));
44     );
45     print("Greedy fast: ", fraction, " = ", printEgypFrac(result));
46     return(Vec(result));
47 }

```

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

### 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. *summands* enthält die Summanden der Zweierpotenzen, die in Summe den Zähler des aktuell betrachteten Bruchs ergeben. *result* ist die Liste der Summanden des Ägyptischen Bruchs.

```

1  binary_algo(fraction)={
2    local(p,q,r,s,Nk,summands,result);
3    result = List();
4    p = numerator(fraction);
5    q = denominator(fraction);
6    Nk=1;
7    while(q > Nk, Nk*=2);
8    if(q == Nk,
9      summands = findDivisorsOf_k_addingup_n(Nk,p);
10     for(i=1, #summands, listput(result, 1/(Nk/summands[i]))),
11     /*else*/
12     [s,r] = divrem(p*Nk,q);
```

6 Anhang

---

```
13 summands = findDivisorsOf_k_addingup_n(Nk,s);
14 for (i=1, #summands, listput(result, 1/(Nk/summands[i])));
15 summands = findDivisorsOf_k_addingup_n(Nk,r);
16 for (i=1, #summands, listput(result, 1/((q*Nk)/summands[i])));
17 );
18 print("Binary: ", fraction, " = ", printEgypFrac(result));
19 return(reverse_vecsort(Vec(result)));
20 }
```

## 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
- [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
- [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