1 Lösungsidee

Die Kernidee des Algorithmus besteht darin, alle Terme zu ermitteln, die durch n-fache Verwendung der gegebenen Ziffer und deren Verknüpfung mit sich selbst (wie in der Aufgabenstellung beschrieben) optimal ihren Wert darstellen. Dies wird induktiv gelöst, wobei für die Errechnung der optimalen Terme mit Größe n alle kleineren Terme vorausgesetzt werden. Zuerst werden alle infrage kommenden Terme mit Größe n gebildet. Hierzu werden alle Terme der Größe i mit allen Termen der Größe $n-i$ unter Verwendung aller binärer Operationen gekreuzt. Daraufhin werden alle Terme entfernt, deren Wert schon mit kleineren oder gleich großen Termen dargestellt wurde. Um mit diesem Verfahren nun die Aufgabe zu lösen, ermittelt man solange größere Terme, bis der gesuchte Wert optimal von einem Term dargestellt wird. Diese Vorangehensweise erlaubt es auch, mit der gleichen Ziffer Repräsentationen für mehrere Zahlen auf einmal zu finden, ohne dass sich die Laufzeit addiert. Hierfür werden einfach größere und größere Terme ermittelt, bis alle Zahlen eine optimale Repräsentation erhalten haben. Die Laufzeit, um mehrere Zahlen zu repräsentieren entspricht hierbei einfach der längsten Laufzeit, bei dem jede Zahl individuell durch den Algorithmus läuft.

Auf ersten Blick scheint dieser Algorithmus eine exponentielle Komplexität zu besitzen, tatsächlich ist dem aber gar nicht so. Im Folgenden wird die Komplexität dieses Verfahrens genauer betrachtet.

1.1 Größenmaß-Funktion

Zur Berechnung der Komplexität wird die Größenmaß-Funktion $\Phi_d^B : \mathbb{Q} \to \mathbb{N}$ betrachtet, die die Anzahl an Instanzen der Ziffer d errechnet, die für die optimale Repräsentation beliebiger rationaler Zahlen benötigt werden. Dabei ist eine Konkatenierung bezüglich der Basis B möglich. Beispielsweise wird die Zahl 11 aus zwei mal der Ziffer 1 zur Basis $B = 10$ generiert. Für die Größenmaß-Funktion gilt insbesondere $\forall d, B : \ \Phi_d^B(d) = 1$.

Lemma 1.1 (Größenmaß für nicht-negative ganze Zahlen) *Für jede nicht-negative ganze Zahl n gilt:*

$$\Phi_d^B(n) \leq n + 2$$

Beweis. — **Fall 1:** $n > 0$

$n > 0$ lässt sich als n-mal wiederholte Summierung von d darstellen:

$$n = \frac{d + d + \ldots + d}{d} \implies \Phi_d^B(n) \leq n + 1 < n + 2$$

Fall 2: $n = 0$

$n = 0$ lässt sich immer mit genau zwei Ziffern darstellen:

$$n = d - d \implies \Phi_d^B(n) \leq 2 = n + 2$$

Corollary 1.1.1 (Größenmaß für ganze Zahlen) *Für jede ganze Zahl n gilt:*

$$\forall d, B : \ \Phi_d^B(n) \leq |n| + 3$$

Beweis. — **Fall 1:** $n \geq 0$

Laut Lemma 1.1 gilt:

$$\Phi_d^B(n) \leq n + 2 = |n| + 2 < |n| + 3$$

Fall 2: $n < 0$

$n < 0$ lässt sich als Differenz aus $0 = d - d$ und $|n| > 0$ darstellen:

$$n = d - d - \frac{d + d + \ldots + d}{d} \implies \Phi_d^B(n) \leq |n| + 3$$

oder alternativ unter Verwendung von unärer Negation:

$$n = -\frac{d + d + \ldots + d}{d} \implies \Phi_d^B(n) \leq |n| + 1 < |n| + 3$$

Lemma 1.2 (Größenmaß für binäre Operationen) *Für jeden binären Operator* $\circ : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ *gilt für alle* $a \in \mathbb{Q},\ b \in \mathbb{Q}$*:*

$$\Phi_d^B(a \circ b) \leq \Phi_d^B(a) + \Phi_d^B(b)$$

Die Anwendung der Größenmaß-Funktion auf binäre Operationen lässt sich also immer durch Addition der Größenmaße der Operanden abschätzen.

Beweis. — Die rationalen Zahlen a und b lassen sich mit $\Phi_d^B(a)$ bzw. $\Phi_d^B(b)$ Ziffern ausdrücken, also lässt sich $a \circ b$ durch Anwendung der Operation $\circ$ auf die Terme für a und b mit $\Phi_d^B(a) + \Phi_d^B(b)$ oder weniger Ziffern ausdrücken.

Corollary 1.2.1 (Größenmaß für rationale Zahlen) *Für jede rationale Zahl* $\frac{a}{b} \in \mathbb{Q},\ a \in \mathbb{Z},\ b \in \mathbb{Z}^+$ *gilt:*

$$\Phi_d^B(\frac{a}{b}) \leq \Phi_d^B(a) + \Phi_d^B(b)$$

Beweis. — $\frac{a}{b}$ ist das Ergebnis der Anwendung des binären Divisions-Operators auf a und b, somit lässt es sich laut Lemma 1.2 mit $\Phi_d^B(a) + \Phi_d^B(b)$ oder weniger Ziffern ausdrücken.

Theorem 1.3 (Genauere obere Grenze des Größenmaßes für positive ganze Zahlen) *Für jede positive ganze Zahl n gilt:*

$$\Phi_d^B(n) \leq \log_d(n) \cdot (d + 4)$$

Beweis. — Jede positive ganze Zahl lässt sich im Horner-Schema darstellen. Betrachte man die natürliche Zahl n in Basis d.

$$n = a_1 + d(a_2 + d(\ldots da_k)),\ a_i \in \mathbb{N},\ 0 \leq a_i < d \tag{1}$$

Hierbei ist es anzumerken, dass keine Basis 1 existiert und dies somit für $d = 1$ nicht anwendbar ist. Im folgenden wird zuerst der generelle Fall betrachtet. Aus Gleichung 1 lässt sich ablesen:

$$\begin{aligned}
\Phi_d^B(n) &\leq \Phi_d^B(a_1 + d(a_2 + d(\ldots da_k))) && \mid 1.2. \\
&\leq \Phi_d^B(a_1) + \Phi_d^B(d) + \Phi_d^B(a_2) + \Phi_d^B(d) + \ldots + \Phi_d^B(d) + \Phi_d^B(a_k) \\
&= (k-1) \cdot \Phi_d^B(d) + \sum_{i=1}^{k} \Phi_d^B(a_i) && (2)
\end{aligned}$$

Da $0 \leq a_i < d$ gilt, folgt mit Lemma 1.1 für jedes a_i: $\Phi_d^B(a_i) \leq d + 2$. Zusammen mit $\Phi_d^B(d) = 1$ folgt aus Abschätzung 2:

$$\begin{aligned}
\Phi_d^B(n) &\leq (k-1) \cdot 1 + \sum_{i=1}^{k} (d+2) \\
&\leq (k-1) + k \cdot (d+2) \\
&\leq k + k \cdot (d+2) = k \cdot (d+3)
\end{aligned}$$

Bei der Repräsentation mit dem Horner-Schema gilt $k = \lfloor \log_d(n) \rfloor$, somit folgt:

$$\Phi_d^B(n) \leq k \cdot (d+3) = \lfloor \log_d(n) \rfloor \cdot (d+3) \leq \log_d(n) \cdot (d+3)$$

Das Problem der non-existenten Basis für den Spezialfall $d = 1$ kann durch den folgenden Trick umgangen werden: Anstelle von $d = 1$ wird die Basis $2 = 1 + 1 = d + d$ betrachtet. In Abschätzung 2 muss dann nur $\Phi_d^B(d)$ durch $\Phi_d^B(d+d) = 2 \cdot \Phi_d^B(d)$ ersetzt werden:

$$\Phi_d^B(n) \leq (k-1) \cdot 2 \cdot \Phi_d^B(d) + \sum_{i=1}^{k} \Phi_d^B(a_i)$$

und es folgt analog zu oben:

$$\begin{aligned} \Phi_d^B(n) &\leq k \cdot (d+4) \\ &\leq \log_d(n) \cdot (d+4) \end{aligned}$$

Somit gilt sowohl im generellen Fall als auch im Spezialfall die Abschätzung $\Phi_d^B(n) \leq \log_d(n) \cdot (d+4)$.

Corollary 1.3.1 (Genauere obere Grenze des Größenmaßes für ganze Zahlen) *Für jede ganze Zahl n gilt:*

$$\Phi_d^B(n) \leq \log_d(n) \cdot (d+4) + 2$$

Beweis. — Analog zum Beweis von Lemma 1.1.1.

1.2 Anzahl an Termen

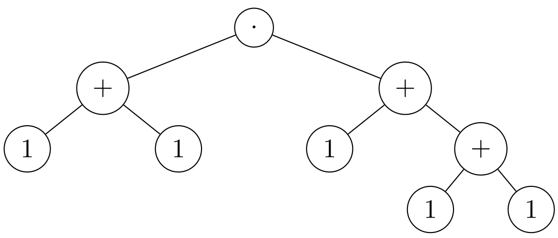
Die Anzahl an Termen mit n Ziffern sei definiert als t_n. Bei Größe 0 gibt es beispielsweise $t_0 = 0$ Terme, bei 1 gibt es $t_1 = 1$ Terme und bei 2 gibt es $t_2 = 6$ Terme ($d; dd; d+d; d-d; d \cdot d; d \div d$).

Theorem 1.4 (Abschätzung für t_n) *Die Anzahl an Termen lässt sich wie folgt nach oben abschätzen*

$$t_n \leq C_n \cdot 5^{n-1},$$

wobei C_n die Catalan Zahlen bezeichnet.

Beweis. — Terme von Größe n können als volle Binärbäume mit n Blättern aufgefasst werden, wobei jedes Blatt eine Ziffer oder eine Wiederholung der Ziffer darstellt und jeder Knoten eine binäre Operation zwischen seinen zwei Kindern darstellt. Am Beispiel für $6 = (1+1) \cdot (1 + (1+1))$:



Hierbei werden auch invalide Terme mit Division durch Null wie $1 \div (1-1)$ mit betrachtet. Jedoch sind alle validen Terme wie oben repräsentierbar, so dass die Anzahl solcher Binärbäume größer als die Anzahl valider Terme ist und somit geeignet für eine obere Abschätzung. Nachfolgend werden zunächst nur Terme ohne Konkatenierung von Ziffern betrachtet.

Betrachte man nun zuerst die Anzahl solcher Binärbäume. Die Anzahl an vollen Binärbäumen mit n Blättern wird durch die Catalan Zahlen C_n beschrieben. Die Catalan Zahlen beschreiben jedoch nur die Anzahl an Binärbäumen, nicht die Anzahl an Binärbäumen mit Operatoren in den Knoten. In einem vollen Binärbaum mit n Blättern gibt es $n-1$ Knoten bzw. $n-1$ Operatoren. Bei 4 Operatoren heißt das, dass die Anzahl an Kombinationen von Operatoren 4^{n-1} beträgt. Folglich ist die Anzahl an Binärbäumen mit Operatoren in den Knoten das Produkt dieser beiden Zahlen: $C_n \cdot 4^{n-1}$.

Um nun auch die Konkatenierung von Ziffern zu berücksichtigen, kann man sich einen Konkatenierungs-Operator vorstellen:

$$l \circ r = B \cdot l + r\,, \qquad \text{bspw.: } 1 \circ 1 = B + 1 = 10 + 1 = 11$$

Dieser würde natürlich auch "unerlaubte" Konkatenierungen erlauben, beispielsweise für $B = 10$:

$$(4+4+4) \circ (4 \cdot 4) = 12 \circ 16 = 12B + 16 = 120 + 16 = 136$$

Jedoch werden so alle validen Konkatenierungen und mehr abgebildet, weshalb er für eine Abschätzung nach oben geeignet ist. Nun lässt sich die vorherige Herangehensweise analog durchführen, jedoch diesmal mit 5 Operatoren. Somit ist die Anzahl an Termen der Größe n kleiner gleich $C_n \cdot 5^{n-1}$.

Theorem 1.5 (Anzahl zu betrachtender Terme) *Aus Corollary 1.3.1, 1.2.1 und Theorem 1.4 folgt, dass für jede rationale Zahl $n \in \mathbb{Q}$ gilt:*

$$\sum_{i=0}^{\Phi_d^B(n)} t_i \in \mathcal{O}(n^\alpha), \qquad \textit{mit } \alpha = \frac{d+4}{\log_{20}(d)}$$

Was bedeutet, dass die Anzahl an Termen, die zur Darstellung von n die Ziffer minimal verwenden, polynomial zu n abschätzbar ist.

Beweis. — Laut Theorem 1.4 gilt:

$$t_n \leq C_n \cdot 5^{n-1}$$

Für die Catalan Zahlen gilt folgende obere Abschätzung[1]:

$$C_n < \frac{4^n}{(n+1)\sqrt{\pi n}}$$

Somit folgt:

$$t_n \leq C_n \cdot 5^{n-1} < \frac{4^n}{(n+1)\sqrt{\pi n}} 5^{n-1} = \frac{4^n \cdot 5^n}{5(n+1)\sqrt{\pi n}} = \frac{20^n}{5(n+1)\sqrt{\pi n}}$$

Nun kann man um die Anzahl der für n zu betrachtender Terme $\Phi_d^B(n)$ statt n einsetzen, um die Anzahl aller Terme zu bestimmen, die genauso viele Ziffern wie die optimale Repräsentation von n haben. Mit Corollary 1.3.1 gilt nach einer ganzen Reihe von Umformungen (siehe Anhang):

$$t_{\Phi_d^B(n)} = \mathcal{O}\Big(n^{\frac{d+4}{\log_{20}(d)}} \cdot \underbrace{\left(\log_d(n) \cdot d\right)^{-\frac{3}{2}}}_{\text{streng monoton fallend}}\Big) = \mathcal{O}(n^\alpha), \qquad \text{mit } \alpha = \frac{d+4}{\log_{20}(d)} \tag{3}$$

Da die Ziffer d dabei immer konstant ist, ist auch α konstant. Somit ist $t_{\Phi_d^B(n)}$ nach oben polynomial abschätzbar.

[1] Siehe Seite 212: https://www.sciencedirect.com/science/article/pii/S0195669886800245

Da der Algorithmus induktiv arbeitet, werden zur Bestimmung der Terme von Größe n die Terme von Größe $n-1$ benötigt. Daher entspricht die Anzahl der Terme, die für eine optimale Repräsentation von n zu betrachten sind, der Summe der Anzahl aller Terme mit genauso vielen oder weniger Ziffern wie die optimale Repräsentation n, also $\sum_{i=0}^{\Phi_d^B(n)} t_i$. Da die Komplexitätsklasse einer Summe von Elementen der polynomialen Komplexitätsklasse wieder die polynomiale Komplexitätsklasse ist, ist diese Summe somit auch polynomial:

$$\sum_{i=0}^{\Phi_d^B(n)} t_i \in \mathcal{O}(n^\alpha), \qquad \text{mit } \alpha = \frac{d+4}{\log_{20}(d)}$$

Da immer alle Ziffern (außer 0) verwendet werden, muss diese Komplexität für alle Ziffern addiert werden. Somit ist die Komplexität eine Summe aus Potenzen, also ebenfalls eine Potenz:

$$t_{\Phi_d^B(n)} \in \mathcal{O}(n^\alpha), \qquad \text{mit } \alpha = \max_{0<d\leq 9} \frac{d+4}{\log_{20}(d)}$$

Auch wenn quasi purer Brute-Force verwendet wird, so belegen diese Überlegungen dennoch, dass eine polynomiale Komplexität vorliegt. Da der eigentliche Algorithmus durch die Eliminierung von Duplikaten noch einmal stärker optimiert ist, ist dessen Komplexität natürlich auch polynomial.

1.3 Exponenten und Fakultäten

Um den Algorithmus auf Exponenten und Fakultäten zu erweitern, kann zuerst Exponentiation als zusätzlicher binärer Operator wie Addition und Multiplikation hinzugefügt werden. Dabei bleibt die Komplexität analog zu vorher polynomial, nur zu einem größeren Exponenten.

Für die Hinzunahme von Fakultäten ist dies jedoch nicht so leicht. Zunächst lässt sich Fakultät relativ leicht in den Algorithmus einbauen, wobei jeder neue optimale Term noch einmal mit einer Fakultät hinten zu der Liste aller optimalen Terme addiert wird. Das Problem besteht allerdings darin, dass Fakultät eine unäre Operation ist, weshalb sie unabhängig von der Anzahl an Instanzen der Ziffer beliebig oft verwendbar ist; beispielsweise ist $((((d!)!)!)!)!$ in der Theorie erlaubt. Betrachtet man jedoch die Werte der Fakultäts-Funktion, so fällt auf, dass dort sehr schnell immens große Werte entstehen; beispielsweise ist $(5!)! = 120!$, was eine Zahl mit 199 Ziffern ist. Dass diese Werte nur sehr selten brauchbar sind, ist relativ offensichtlich.

Zur Umgehung dieses Problems habe ich daher ein Limit eingefügt, das Fakultäten von Zahlen größer 20 verbietet, wobei 20! die größte durch signierte 64-Bit Integer darsellbare Fakultät ist. Somit ist $(4!)!$ gerade noch erlaubt, jedoch $(5!)!$ verboten. Dies lässt sich auch leicht implementieren, wobei jeder neue Term mit Wert $x < 80$ den neuen Term $x!$ rekursiv einfügt. Dabei muss man jedoch noch darauf achten, dass keine Endlosschleifen auftauchen: $1 = 1! = \ldots = ((1!)!)!$

Es lässt sich erkennen, dass hierbei höchstens zwei Fakultäten in Reihe auftreten können. Das bedeutet, dass für jeden neu gefundenen optimalen Term noch (höchstens) zwei weitere Terme addiert werden. Dies bedeutet, dass sich die Rechenzeit (höchstens) verdreifacht, und somit immer noch polynomial ist.

Im Hinblick auf Fakultäten ist noch bemerkenswert, dass nun auch die Ziffer 0 verwendet werden kann: $0! = 1$. Die Beispiele unten enthalten daher auch Lösungen mit der Ziffer 0.

2 Umsetzung

Der Algorithmus wurde in C# 8.0 mit .NET Core 3.1 implementiert. Es wurde als Library CommandLineParser[2] verwendet. Diese implementiert Methoden um Unix-Style Konsolenargumente entgegenzunehmen. Der Code ist in zwei Projekte geteilt; `Afg2Geburtstag.CLI` und `Afg2Geburtstag`. `Afg2Geburtstag.CLI` kümmert sich um das Konsolen-Interface und ruft `Afg2Geburtstag` auf, was den eigentlichen Algorithmus enthält. `Afg2Geburtstag` definiert einige Typen:

[2] https://github.com/commandlineparser/commandline

Rational definiert rationale Zahlen und Operationen auf diesen.

ITerm repräsentiert eine Schnittstelle für beliebige Terme mit einem `Rational` Wert. Wird von `Rational` implementiert.

UnaryOperator repräsentiert einen unären Operator der zur Erstellung einer `UnaryOperation` verwendet werden kann.

UnaryOperation repräsentiert eine unäre Operation auf einem `ITerm` und implementiert selbst `ITerm`.

BinaryOperator repräsentiert einen binären Operator der zur Erstellung einer `BinaryOperation` verwendet werden kann.

BinaryOperation repräsentiert eine binäre Operation zwischen zwei `ITerms` und implementiert selbst `ITerm`.

DigitRepresenter enthält den Kernalgorithmus für eine Ziffer und mehrere Zielzahlen.

`UnaryOperation` und `BinaryOperation` speichern als Optimierung ihren berechneten Wert und Hashcode. Weiterhin sind die Operanden der beiden Typen `ITerm` Instanzen und somit Referenztypen. Dies bedeutet, dass jede Operation nur einmal gespeichert wird und nicht in anderen Operationen, die sie verwenden, gedoppelt wird.

In `DigitRepresenter`, worin der Hauptalgorithmus implementiert wird, werden an vielen Stellen Hashsets verwendet, um schnelle Lookups zu erlauben. Spezifischer, da `HashSet<T>` nicht Thread-Safe ist, verwendet der Code `ConcurrentDictionary<T, byte>`, wobei der `byte`-Wert ignoriert wird.

Für Informationen zur Verwendung des Programms kann `Afg2Geburtstag.exe --help` ausgeführt werden.

3 Beispiele

3.1 Ohne Exponenten und Fakultäten

Argumente: `--targets 2020, 2030, 2080, 2980 --digits 1,2,3,4,5,6,7,8,9 --sync --latex`

Digit	Value	Term	Digit Usages	Time
1	2020	$\left((1+1)\cdot\left(\frac{(11111-1)}{11}\right)\right)$	10	0.092s
1	2020	$\left(\frac{((1+1)\cdot(11111-1))}{11}\right)$	10	0.092s
1	2080	$\left(\left(1+\left(((1+1)\cdot(1-111))+11\right)\right)\cdot(1-11)\right)$	12	0.124s
1	2030	$\left((11-1)\cdot\left(1+\left((1+1)\cdot(1-(11-111))\right)\right)\right)$	12	0.126s
2	2020	$\left(2\cdot\left(\left((2+(2\cdot 22))\cdot 22\right)-2\right)\right)$	8	0.008s
2	2080	$\left(\left(2\cdot(2\cdot(22-2))\right)\cdot(2+(2+22))\right)$	9	0.010s
2	2030	$\left(\left(2\cdot\left(2+\left((2+(2\cdot 22))\cdot 22\right)\right)\right)+2\right)$	9	0.024s
2	2980	$\left(\left(2-\left((2+22)\cdot\left((2\cdot(2-22))-22\right)\right)\right)\cdot 2\right)$	11	0.133s
3	2020	$\left(((3+333)\cdot(3+3))+\left(3+\left(\frac{3}{3}\right)\right)\right)$	9	0.009s

3	2080	$\left(\left(33 \cdot (33 + (33 - 3))\right) + \left(\frac{3}{3}\right)\right)$	9	0.014s
3	2030	$\left(33 - \left(\left(\frac{3}{3}\right) - (333 \cdot (3 + 3))\right)\right)$	9	0.015s
3	2980	$\left(\left(\frac{3}{3}\right) + \left(3 \cdot \left(3 + (33 \cdot (33 - 3))\right)\right)\right)$	9	0.026s
4	2080	$\left((44 - 4) \cdot (4 + (4 + 44))\right)$	7	0.002s
4	2020	$\left(4 - \left((4 + 4) \cdot \left(4 - \left(4 \cdot (4 \cdot (4 \cdot 4))\right)\right)\right)\right)$	8	0.004s
4	2980	$\left(4 - \left((4 \cdot 4) - \left(\left(4 + (4 \cdot (4 \cdot 4))\right) \cdot 44\right)\right)\right)$	9	0.009s
4	2030	$\left(\left(4 \cdot \left(\left(4 \cdot \left(4 \cdot (4 \cdot (4 + 4))\right)\right) - 4\right)\right) - \left(\frac{(4+4)}{4}\right)\right)$	10	0.192s
5	2080	$\left(\left(5 + \left(\frac{55}{5}\right)\right) \cdot \left(5 + (5 \cdot (5 \cdot 5))\right)\right)$	8	0.004s
5	2980	$\left(\left(5 + (5 + (55 \cdot 55))\right) - 55\right)$	8	0.004s
5	2030	$\left(\left(5 \cdot \left(5 + \left(((5 \cdot 5) + 55) \cdot 5\right)\right)\right) + 5\right)$	8	0.007s
5	2020	$\left(\left(5 \cdot \left(5 + \left(((5 \cdot 5) + 55) \cdot 5\right)\right)\right) - 5\right)$	8	0.007s
6	2080	$\left(\left(\left(\frac{(6+6)}{6}\right) + ((6 \cdot 6) - 6)\right) \cdot \left(66 - \left(\frac{6}{6}\right)\right)\right)$	10	0.068s
6	2030	$\left(\left(\left(\frac{6}{6}\right) - 6\right) \cdot \left(\left(\frac{(6-66)}{6}\right) - (6 \cdot 66)\right)\right)$	10	0.073s
6	2020	$\left(\left(6 - \left(\frac{6}{6}\right)\right) \cdot \left(\left(\frac{(6+6)}{6}\right) + (6 + (6 \cdot 66))\right)\right)$	10	0.132s
6	2980	$\left(\left(\left(6 + \left(\frac{6}{6}\right)\right) \cdot \left(\left(6 \cdot (6 \cdot (6 + 6))\right) - 6\right)\right) - \left(\frac{(6+6)}{6}\right)\right)$	11	0.466s
7	2030	$\left(\left(7 \cdot \left(7 + \left(7 \cdot ((7 \cdot 7) - 7)\right)\right)\right) - 77\right)$	8	0.013s
7	2020	$\left(\left(\frac{77}{7}\right) + \left(7 \cdot \left(\left(7 \cdot ((7 \cdot 7) - 7)\right) - 7\right)\right)\right)$	9	0.015s
7	2080	$\left(\frac{\left(7 + \left(77 \cdot \left(((7+7) \cdot (7+7)) - 7\right)\right)\right)}{7}\right)$	9	0.029s
7	2980	$\left(\left(\left(7 \cdot \left(7 + \left(77 + (7 \cdot (7 \cdot 7))\right)\right)\right) - \left(\frac{(7+7)}{7}\right)\right) - 7\right)$	11	0.505s
8	2080	$\left(\left(8 + (8 + (8 + 8))\right) \cdot \left(\left(\frac{8}{8}\right) + (8 \cdot 8)\right)\right)$	8	0.004s

8	2020	$\left(\frac{\left(8-\left(8\cdot\left(8-\left(8\cdot(8\cdot8)\right)\right)\right)\right)}{\left(\frac{(8+8)}{8}\right)}\right)$	9	0.023s
8	2030	$\left(\left(\frac{(8-88)}{8}\right)-\left(8-\left(8\cdot\left((8+8)\cdot(8+8)\right)\right)\right)\right)$	10	0.106s
8	2980	$\left(\left(\frac{8}{8}\right)+\left(\frac{\left(8+\left(\left(8\cdot\left(8\cdot(8\cdot8)\right)\right)\cdot(8\cdot8)\right)\right)}{88}\right)\right)$	11	0.345s
9	2080	$\left(\left(\frac{9}{9}\right)+\left(\left(9+\left(\frac{(9+99)}{9}\right)\right)\cdot99\right)\right)$	9	0.014s
9	2020	$\left(\frac{\left(\left(\left(\frac{99}{9}\right)+999\right)\cdot(9+9)\right)}{9}\right)$	9	0.016s
9	2980	$\left(\left(\frac{9}{9}\right)+\left(\left(\left(9\cdot\left(9+\left((9+9)\cdot(9+9)\right)\right)\right)-9\right)-9\right)\right)$	10	0.049s
9	2030	$\left(\left((9\cdot9)-\left(\frac{99}{9}\right)\right)\cdot\left(9+\left(9+\left(\frac{99}{9}\right)\right)\right)\right)$	10	0.088s

3.2 Mit Exponenten und Fakultäten

Argumente: `--targets 2020, 2030, 2080, 2980 --digits 0,1,2,3,4,5,6,7,8,9 --exponentiation --factorial --sync --latex`

Digit	**Value**	**Term**	**Digit Usages**	**Time**
0	2980	$\left(\left((0!)+\left((0!)+\left((0!)+(0!)\right)\right)\right)\cdot\left((0!)+\left(((((0!)+(0!))+((0!)+(0!)))!)+(((((0!)+(0!))+(0!))!)!)\right)\right)\right)$	24	0.476s
0	2080	$\left(\left(((0!)+(0!))^{(((((0!)+(0!))+(0!))!)-(0!))}\right)\cdot\left((0!)+\left(((0!)+(0!))^{((((0!)+(0!))+(0!))!)}\right)\right)\right)$	24	0.487s
0	2020	$\left(((0!)+(0!))\cdot\left((0!)+\left((0!)-\left(\frac{\left(((0!)+((((0!)+(0!))+(0!))!))!\right)}{\left((0!)-\left((((0!)+(0!))+(0!))!\right)\right)}\right)\right)\right)\right)$	24	0.490s

0	2030	$\left((0!) + \left(\left(\left((0!) + ((0!) + (0!))\right) \cdot \left(((0!) + ((0!) + ((((0!) + (0!)) + ((0!) + (0!)))!)))^{((0!)+(0!))}\right)\right) + (0!)\right)\right)$	26	0.838s
1	2080	$\left(\left((1 + (1 + 1)) \cdot 11\right) + \left(\left((1 + 1)^{11}\right) - 1\right)\right)$	10	0.037s
1	2030	$\left(\left(\left(\left((1 + 1)^{11}\right) - 1\right) - (((1 + 1) + 1)!)\right) - 11\right)$	10	0.037s
1	2020	$\left(\frac{((11111-1)\cdot(1+1))}{11}\right)$	10	0.047s
1	2980	$\left(\left(((((1 + 1) + 1)!) + 1)!\right) - \left(\left((1 + 1)^{11}\right) + 11\right)\right) - 1\right)$	11	0.180s
2	2080	$\left(\left((2 * 22)^2\right) + \left((((2 * 2)!)/2)^2\right)\right)$	8	0.019s
2	2030	$\left(\left(\left(2^{(22/2)}\right) - 22\right) + (2 + 2)\right)$	8	0.020s
2	2020	$\left(\left(\left(2^{(22/2)}\right) - (2 + 2)\right) - ((2 * 2)!)\right)$	8	0.039s
2	2980	$\left(\left((2 * ((2 * 2)!))^2\right) + \left((2 + ((2 * 2)!))^2\right)\right)$	8	0.043s
3	2080	$\left(\left(3 \cdot \left((((3 * 3) - 3)!) - \left(3^3\right)\right)\right) + \left(\frac{3}{3}\right)\right)$	8	0.077s
3	2030	$\left(\left(33 - \left(\frac{3}{3}\right)\right) + (333 \cdot (3 + 3))\right)$	9	0.188s
3	2020	$\left(\left(3 + ((3 + 3) \cdot (3 + 333))\right) + \left(\frac{3}{3}\right)\right)$	9	0.326s
3	2980	$\left(\left(\frac{3}{3}\right) - \left(3 \cdot \left(((3 - 33) \cdot 33) - 3\right)\right)\right)$	9	0.386s
4	2080	$\left((4 + 4) \cdot \left(4 + \left(4^4\right)\right)\right)$	5	0.001s
4	2020	$\left(4 + \left(\left(\left(4^4\right) - 4\right) \cdot (4 + 4)\right)\right)$	6	0.002s
4	2980	$\left(\left(\left(\left(4^4\right) - 4\right) \cdot (4 + (4 + 4))\right) - 44\right)$	8	0.052s
4	2030	$\left(\left(4^4\right) - \left(\left(\frac{(4+4)}{4}\right) - (4 \cdot 444)\right)\right)$	9	0.174s
5	2980	$\left(\left(5 - \left(5 \cdot (5 + (5 \cdot 5))\right)\right) + \left(5^5\right)\right)$	7	0.005s
5	2080	$\left(\left(5 + (5 \cdot (5 \cdot 5))\right) \cdot \left(5 + \left(\frac{55}{5}\right)\right)\right)$	8	0.035s
5	2030	$\left(\left(5 + \left(5^5\right)\right) - \left(((5 \cdot 5) - 5) \cdot 55\right)\right)$	8	0.036s
5	2020	$\left(\left(5 \cdot \left(5 + \left(5 \cdot ((5 \cdot 5) + 55)\right)\right)\right) - 5\right)$	8	0.062s

6	2030	$\left(\left(\left(\frac{6}{6}\right)-(6\cdot6)\right)\cdot\left(6-\left(((6+6)/6)^{6}\right)\right)\right)$	9	0.232s
6	2080	$\left(\left(\left(\frac{6}{6}\right)-66\right)\cdot\left((6-(6\cdot6))-\left(\frac{(6+6)}{6}\right)\right)\right)$	10	0.721s
6	2020	$\left(\frac{\left(\left((66\cdot66)-\left(6-\left(\frac{\left(6^{6}\right)}{6}\right)\right)\right)-6\right)}{6}\right)$	10	1.284s
6	2980	$\left(\left(((6+6)/6)^{6}\right)+\left((6+(6-66))^{((6+6)/6)}\right)\right)$	11	2.073s
7	2030	$\left(\left(7\cdot\left(7-\left(7\cdot(7-(7\cdot7))\right)\right)\right)-77\right)$	8	0.041s
7	2020	$\left(\left(\frac{77}{7}\right)-\left(7\cdot\left(7+\left(7\cdot(7-(7\cdot7))\right)\right)\right)\right)$	9	0.239s
7	2080	$\left(\frac{\left(7+\left(77\cdot\left(((7+7)\cdot(7+7))-7\right)\right)\right)}{7}\right)$	9	0.372s
7	2980	$\left(\left(\left(\left(\frac{77}{7}\right)-7\right)\cdot777\right)-\left(((7+7)/7)^{7}\right)\right)$	11	2.234s
8	2080	$\left(\left(\left(\frac{8}{8}\right)+(8\cdot8)\right)\cdot\left(8+(8+(8+8))\right)\right)$	8	0.033s
8	2980	$\left(\left(\frac{8}{8}\right)+\left(\frac{\left(8+\left(\frac{\left(8^{8}\right)}{(8\cdot8)}\right)\right)}{88}\right)\right)$	9	0.098s
8	2020	$\left(\frac{\left(8+\left(8\cdot\left((8\cdot(8\cdot8))-8\right)\right)\right)}{\left(\frac{(8+8)}{8}\right)}\right)$	9	0.228s
8	2030	$\left(\left(\frac{(8-88)}{8}\right)-\left(8-\left(8\cdot((8+8)\cdot(8+8))\right)\right)\right)$	10	0.390s
9	2030	$\left(\left(((9+9)/9)^{(99/9)}\right)-(9+9)\right)$	8	0.012s
9	2080	$\left(\left(9+\left(9+\left(9-\left(\frac{9}{9}\right)\right)\right)\right)\cdot\left((9\cdot9)-\left(\frac{9}{9}\right)\right)\right)$	9	0.051s
9	2020	$\left(\left(\left(\frac{99}{9}\right)+999\right)\cdot\left(\frac{(9+9)}{9}\right)\right)$	9	0.053s
9	2980	$\left(\left(\frac{9}{9}\right)+\left(\left(\left(9+(9\cdot(9+9))\right)\cdot(9+9)\right)-99\right)\right)$	10	0.259s

4 Code

```
/// <summary>
/// Represents an arbitrary mathematical teerm.
/// </summary>
public interface ITerm
{
    /// <summary>
    /// The value of the term.
    /// </summary>
    Rational Value { get; }

    /// <summary>
    /// Returns a string that represents the current object.
    /// </summary>
    /// <returns>A string that represents the current object.</returns>
    string ToString();

    /// <summary>
    /// Returns a string that represents the current object as latex code.
    /// </summary>
    /// <returns>A string that represents the current object as latex code.</returns>
    string ToLaTeX();
}
```

```
using RationalSet = System.Collections.Concurrent.ConcurrentDictionary<Rational, byte>;
using TermSet = System.Collections.Concurrent.ConcurrentDictionary<ITerm, byte>;

/// <summary>
/// Contains the main algorithm.
/// </summary>
public class DigitRepresenter
{
    /// <summary>
    /// All binary operators available.
    /// </summary>
    public List<BinaryOperator> BinaryOperators { get; }

    /// <summary>
    /// Contains a unary operator. <c>null</c> if no unary operator is to be used.
    /// Multiple unary operators are no supported.
    /// </summary>
    public UnaryOperator? UnaryOperator { get; }

    /// <summary>
    /// All calculated terms, where the set at index i contains all the terms which use the digit i
        times.
    /// </summary>
    public List<TermSet> TermsOfSize { get; }

    /// <summary>
    /// Contains all values of all terms.
    /// </summary>
    public RationalSet AllValues { get; }

    public long Digit { get; }
    public long Base { get; }

    /// <summary>
    /// All targets to find representations for, along with their optimal representation, if one
        was found.
    /// </summary>
    public ConcurrentDictionary<Rational, ITerm?> Targets { get; }

```

```
    /// <summary>
    /// The number of targets for which no optimal representation has been found yet.
    /// </summary>
    public int UnfoundTargets { get; private set; }

    /// <summary>
    /// An action to execute once an optimal representation for any given target was found.
    /// </summary>
    public Action<ITerm, int> OnFound { get; }

    public DigitRepresenter(
        List<BinaryOperator> binaryOperators,
        UnaryOperator? unaryOperator,
        ConcurrentDictionary<Rational, ITerm?> hitTargets,
        Action<ITerm, int> onFound,
        long digit,
        long @base = 10)
    {
        TermsOfSize = new List<TermSet>() { new TermSet() };
        AllValues = new RationalSet();
        BinaryOperators = binaryOperators;
        UnaryOperator = unaryOperator;
        Digit = digit;
        Base = @base;

        Targets = hitTargets;
        UnfoundTargets = hitTargets.Count;
        OnFound = onFound;
    }

    /// <summary>
    /// Calculates all terms of size.
    /// </summary>
    /// <param name="size">The size of the terms to calculate.</param>
    public void CalculateAllOfSize(int size)
    {
        if (TermsOfSize.Count < size) CalculateAllOfSize(size - 1);
        else if (TermsOfSize.Count > size) return;
        var currentTerms = new TermSet();

        // The digit concatenated with itself <size> times
        var repeatedDigit = Digit;
        for (int i = 1; i < size; i++) repeatedDigit = (repeatedDigit * Base) + Digit;
        RegisterTerm(currentTerms, new Rational(repeatedDigit), size);

        // Execute all different ways of combining the smaller terms to terms of size <size> in
            parallel
        Parallel.For(1, size, i =>
        {
            var allLhs = TermsOfSize[i];
            var allRhs = TermsOfSize[size - i];

            foreach (var lhs in allLhs)
            {
                foreach (var rhs in allRhs)
                {
                    foreach (var @operator in BinaryOperators)
                    {
                        var term = BinaryOperation.Create(@operator, lhs.Key, rhs.Key);

                        RegisterTerm(currentTerms, term, size);

                        if (UnfoundTargets == 0) return;
                    }
                }
```

```
            }
        });

        TermsOfSize.Add(currentTerms);
    }

    /// <summary>
    /// Adds <paramref name="term"/> and repeated applications of <see cref="UnaryOperator"/> upon it
    /// to <paramref name="termSet"/>if the terms are optimal representations of their values.
    /// </summary>
    /// <param name="termSet">The set of terms to add the terms to.</param>
    /// <param name="term">The term.</param>
    /// <param name="digitCount">The amount of times the digit is used in <paramref name="term"/>.</param>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    private void RegisterTerm(TermSet termSet, ITerm? term, int digitCount)
    {
        if (term == null) return;
        if (AddTermIfNew(termSet, term, digitCount)) return;

        if (UnaryOperator == null) return;

        // Apply the unary operator repeatedly until either
        // - The value stops changing
        // - The unary operator cannot be applied anymore (values too big, trying to compute (-1)! etc.)
        do
        {
            var newTerm = UnaryOperation.Create(UnaryOperator, term);
            if (newTerm == null || newTerm.Value == term.Value) break;
            term = newTerm;
        }
        while (AddTermIfNew(termSet, term, digitCount));
    }

    /// <summary>
    /// Adds <paramref name="term"/> to <paramref name="termSet"/> if the term are optimal representations of
    /// their values and their values are not yet represented by any other term.
    /// If the term is the value of a target <see cref="OnFound"/> is called
    /// and the term is stored as its optimal representation.
    /// </summary>
    /// <param name="termSet">The set of terms to add the term to.</param>
    /// <param name="term">The term.</param>
    /// <param name="digitCount">The amount of times the digit is used in <paramref name="term"/>.</param>
    /// <returns>Whether the value was added.</returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    private bool AddTermIfNew(TermSet termSet, ITerm? term, int digitCount)
    {
        // Don't add null or exisitng values
        if (term == null || AllValues.ContainsKey(term.Value))
        {
            return false;
        }

        // Check if the terms value is a target
        if (Targets.TryGetValue(term.Value, out var otherTerm) && otherTerm == null)
        {
            Targets[term.Value] = term;
            UnfoundTargets--;
            OnFound(term, digitCount);
        }

```

```
        // Add the term as key with value 0, as the value is ignored
        AllValues.TryAdd(term.Value, 0);
        termSet.TryAdd(term, 0);

        return true;
    }
}
```

5 Anhang

5.1 Rechnung zu Theorem 1.5

Der Beweis von Theorem 1.5 führt auf folgende Abschätzung

$$t_n \leq C_n \cdot 5^{n-1} < \frac{4^n}{(n+1)\sqrt{\pi n}} 5^{n-1} = \frac{20^n}{5(n+1)\sqrt{\pi n}}$$

Nun kann man um die Anzahl der zu betrachtender Terme $\Phi_d^B(n)$ statt n einsetzen, um die Anzahl aller Terme zu bestimmen, die genauso viele Ziffern wie die optimale Repräsentation von n haben. Unter Verwendung der Abschätzung $\Phi_d^B(n) \leq \log_d(n) \cdot (d+4) + 2$ aus Corollary 1.3.1 ergeben sich folgende Umformungen:

$$\begin{aligned}
t_{\Phi_d^B(\frac{a}{b})} &< \frac{20^{\log_d(n)\cdot(d+4)+2}}{5\Big(\big(\log_d(n)\cdot(d+4)+2\big)+1\Big)\cdot\sqrt{\pi\cdot\log_d(n)\cdot(d+4)+2}} \\
&= \frac{(20^{\log_d(n)})^{d+4}\cdot 20^2}{5\Big(\big(\log_d(n)\cdot(d+4)+2\big)+1\Big)\cdot\sqrt{\pi\cdot\log_d(n)\cdot(d+4)+2}} \\
&= 80\cdot\frac{(20^{\log_d(n)})^{d+4}}{\big(\log_d(n)\cdot(d+4)+3\big)\cdot\sqrt{\pi\cdot\log_d(n)\cdot(d+4)+2}} \\
&\leq 80\cdot\frac{(20^{\log_d(n)})^{d+4}}{\log_d(n)\cdot d\cdot\sqrt{\pi\cdot\log_d(n)\cdot d}} \\
&\leq 80\cdot\frac{(20^{\frac{\log_{20}(n)}{\log_{20}(d)}})^{d+4}}{\log_d(n)\cdot d\cdot\sqrt{\log_d(n)\cdot d}} \\
&\leq 80\cdot\frac{(n^{\frac{1}{\log_{20}(d)}})^{d+4}}{\big(\log_d(n)\cdot d\big)^{\frac{3}{2}}} \\
&\leq 80\cdot n^{\frac{d+4}{\log_{20}(d)}}\cdot\big(\log_d(n)\cdot d\big)^{-\frac{3}{2}} \\
&\in \mathcal{O}(n^{\frac{d+4}{\log_{20}(d)}}\cdot\big(\log_d(n)\cdot d\big)^{-\frac{3}{2}})
\end{aligned}$$