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 entspricht hierbei einfach der längsten Laufzeit der individuellen Zahlen.

Auf ersten Blick scheint dieser Algorithmus eine stark explosive 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 wird, wobei Konkatenierung bezüglich der Basis B stattfindet (also z.B. 11 aus zwei mal der Ziffer 1 generieren zur Basis $B = 10$). Für die Größenmaß-Funktion gilt Beispielsweise $\forall d, B : \ \Phi_d^B(d) = 1$.

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 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 d 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 natürlichen 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 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 bei $d = 1$ nicht anwendbar ist. Im folgenden wird zuerst der generelle Fall betrachtet. Aus 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 1.1 für jedes a_i: $\Phi_d^B(a_i) \leq d + 2$. Zusammen mit $\Phi_d^B(d) = 1$ folgt aus 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:

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

Betrachtet man nun noch einmal den Spezialfall $d = 1$: Um das Problem der non-existenten Basis zu umgehen, kann anstatt $d = 1$ die Basis $2 = 1 + 1 = d + d$ betrachtet werden.

Betrachtet man nun 2:

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

Um diese Umgehung des Problems einzubauen, muss nur $\Phi_d^B(d)$ mit $\Phi_d^B(d+d) = 2 \cdot \Phi_d^B(d)$ ersetzt werden:

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

Somit 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 1.1.1.

1.2 Anzahl an Termen

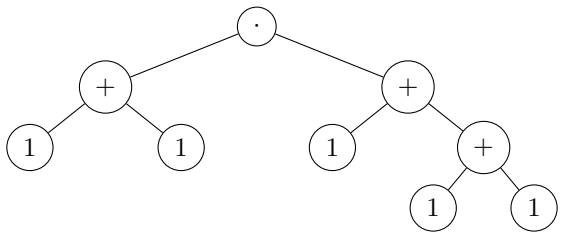
Die Anzahl an Termen mit n Ziffern sei definiert als t_n. Bei Größe 0 gibt es beispielsweise $t_0 = 0$ Terme, bei 1 gibt es $t_1 = 1$ Terme und bei 2 gibt es $t_2 = 6$ Terme $(d; dd; d+d; d-d; d \cdot d; d \div d)$.

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, somit ist das folgende Was ist denn an dieser Stelle das 'folgende'? 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 noch Konkatenierung von Ziffern mitzubetrachten kann man sich einen Konkatenierungs-Operator vorstellen:

$$l \circ r = B \cdot l + r \qquad 1 \circ 1 = B + 1 = 10 + 1 = 11$$

Dieser würde natürlich auch "unerlaubte" Konkatenierungen erlauben:

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

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 1.3.1, 1.2.1 und 1.4 folgt, dass für jede rationale Zahl $n \in \mathbb{Q}$ gilt:*

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

Was bedeuten mag, dass die Anzahl an Termen, die die Ziffer so oft verwenden wie minimal notwendig ist um n darzustellen, respektiv zu n polynomial abschätzbar ist.

Beweis. — Laut 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 $\Phi_d^B(n)$ für n einsetzen. Mit Corollarry 1.3.1 gilt nach einer ganzen Reihe von Umformungen (siehe Anhang):

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

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 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 weiterer binärer Operator wie Addition und Multiplikation hinzugefügt werden, wobei die Komplexität analog zu vorher polynomial bleibt, nur mit einem größeren Exponenten. Für die Integration 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 hierbei ist, 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 immernoch polynomial ist.

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

2 Umsetzung

Der Algorithmus wurde in C# 8.0 mit .NET Core 3.1 implementiert. 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:

BigRational definiert mithilfe des BCL-Typs `BigInteger` einen Typen für (theoretisch) unendlich große rationale Zahlen.

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

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

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

DigitFarm enthält den Kernalgorithmus.

`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 `DigitFarm`, 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.

3 Beispiele

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

Digit	Value	Term	Digit Usages	Time
0	2080	$\left(\left(\left(\left(\left(\left(0!\right)+\left(0!\right)\right)+\left(0!\right)\right)!\right)!\right)\cdot\left(\left(\left(0!\right)+\left(0!\right)\right)-\left(23\left(\left(\left(0!\right)+\left(\left(0!\right)+\left(0!\right)\right)\right)\right)^{\left(0-\left(\left(0!\right)+\left(0!\right)\right)\right)}\right)-\left(0!\right)\right)\right)$	10	0.782s
0	2980	$\left(\left(\left(0!\right)+\left(\left(0!\right)+\left(\left(0!\right)+\left(0!\right)\right)\right)\right)\cdot\left(\left(\left(0!\right)+\left(\left(\left(\left(0!\right)+\left(0!\right)\right)+\left(\left(0!\right)+\left(0!\right)\right)\right)!\right)\right)+\left(\left(\left(\left(0!\right)+\left(0!\right)\right)+\left(\left(0!\right)+\left(0!\right)\right)\right)!\right)\right)$	11	1.312s
0	2020	$\left(\left(\left(0!\right)+\left(0!\right)\right)+\left(\frac{\left(\left(\left(0!\right)+\left(\left(\left(\left(0!\right)+\left(0!\right)\right)+\left(\left(0!\right)+\left(0!\right)\right)\right)!\right)\right)\right)}{\left(\left(\left(\left(\left(0!\right)+\left(0!\right)\right)+\left(0!\right)\right)!\right)-\left(0!\right)\right)}\right)\right)$	24	1.739s
0	2030	$\left(\left(\left(0!\right)+\left(\left(0!\right)+\left(0!\right)\right)\right)\cdot\left(\left(\left(0!\right)+\left(\left(0!\right)+\left(\left(\left(\left(\left(0!\right)+\left(0!\right)\right)+\left(\left(0!\right)+\left(0!\right)\right)\right)!\right)\right)\right)\right)^{\left(\left(0!\right)+\left(0!\right)\right)}\right)+\left(\left(0!\right)\right.$	26	7.082s
1	2020	$\left(\left(\left(1-\left(\frac{1}{11}\right)\right)\cdot 1111\right)\cdot\left(1+1\right)\right)$	10	0.288s
1	2030	$\left(\left(\left(\left(1+1\right)^{11}\right)-\left(1+\left(\left(\left(1+1\right)+1\right)!\right)\right)\right)-11\right)$	10	0.378s
1	2080	$\left(\left(\left(1+1\right)^{\left(\left(\left(\left(1+1\right)+1\right)!\right)-1\right)}\right)+\left(\left(1+1\right)^{11}\right)\right)$	10	0.436s
1	2980	$\left(\left(\left(\left(\left(\left(1+1\right)+1\right)!\right)+1\right)!\right)-\left(\left(1+1\right)^{11}\right)\right)-\left(11+11\right)\right)$	11	2.318s
2	2020	$\left(\left(\left(2*2\right)!\right)^{2}\right)+\left(2\cdot\left(2+\left(\left(\left(2+2\right)+2\right)!\right)\right)\right)$	8	0.065s
2	2030	$\left(\left(\left(2^{\left(22/2\right)}\right)-22\right)+\left(2+2\right)\right)$	8	0.142s
2	2080	$\left(\left(2\cdot\left(22-2\right)\right)\cdot\left(2\cdot\left(2+\left(\left(2*2\right)!\right)\right)\right)\right)$	8	0.143s
2	2980	$\left(\frac{\left(\left(\left(2*2\right)!\right)+\left(2^{\left(2^{\left(2+2\right)}\right)}\right)\right)}{22}\right)$	8	0.179s
3	2080	$\left(\left(\left(3*3\right)-3\right)!\right)\cdot\left(3-\left(\frac{3}{\left(3^{3}\right)}\right)\right)\right)$	7	0.029s
3	2020	$\left(\left(333+\left(\frac{33}{\left(3\cdot 3\right)}\right)\right)\cdot\left(3+3\right)\right)$	9	2.542s
3	2030	$\left(\left(33+\left(333\cdot\left(3+3\right)\right)\right)-\left(\frac{3}{3}\right)\right)$	9	2.546s
3	2980	$\left(\left(\left(3\cdot 33\right)+\left(\frac{3}{\left(3\cdot 3\right)}\right)\right)\cdot\left(3+\left(3^{3}\right)\right)\right)$	9	2.822s

4	2080	$\left(\left(4+\left(4^4\right)\right)\cdot(4+4)\right)$	5	0.008s
4	2020	$\left(\left((4+4)\cdot\left(\left(4^4\right)-4\right)\right)+4\right)$	6	0.012s
4	2980	$\left(\left((4+(4+4))\cdot\left(\left(4^4\right)-4\right)\right)-44\right)$	8	0.282s
4	2030	$\left(\left(\left(\frac{\left(\left((4+4)^4\right)-4\right)}{(4+4)}\right)-4\right)\cdot 4\right)$	8	0.871s
5	2980	$\left(\left(5^5\right)+\left(5-\left(5\cdot(5+(5\cdot 5))\right)\right)\right)$	7	0.018s
5	2020	$\left(\left(5^5\right)+\left(\left((5-(5\cdot 5))\cdot 55\right)-5\right)\right)$	8	0.234s
5	2030	$\left(\left(5^5\right)+\left(\left((5-(5\cdot 5))\cdot 55\right)+5\right)\right)$	8	0.235s
5	2080	$\left(\left(5+(5\cdot(5\cdot 5))\right)\cdot\left(5+\left(\frac{55}{5}\right)\right)\right)$	8	0.360s
6	2030	$\left(\left((6\cdot 6)-\left(\frac{6}{6}\right)\right)\cdot\left(\left(((6+6)/6)^6\right)-6\right)\right)$	9	4.690s
6	2080	$\left(\left(((6+6)/6)^6\right)\cdot\left(\frac{\left((6\cdot 66)-6\right)}{(6+6)}\right)\right)$	10	39.145s
6	2980	$\left(\left(6-\left(\frac{\left(\left(\frac{6}{6}\right)+66\right)}{(6\cdot 6)}\right)\right)\cdot\left((6*(6/6))!\right)\right)$	10	104.364s
6	2020	$\left(\left(6-\left(\frac{\left(6-\left(\left(\frac{\left(6^6\right)}{6}\right)-6\right)\right)}{6}\right)\right)+\left((6*(6/6))!\right)\right)$	10	116.782s

4 Code

TBD

5 Anhang

5.1 Rechnung zu 1.5

Der Beweis von Theorem 1.5 führt auf folgende Abschätzung

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

Hier kann n durch $\Phi_d^B(n)$ ersetzt werden. Unter Verwendung der Abschätzung $\Phi_d^B(n) \leq \log_d(n) \cdot (d+4) + 2$ aus Corollarry 1.3.1 ergeben sich folgende Umformungen:

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