1.2 Der binäre Potenzalgorithmus

Das Verfahren zum möglichst effizienten Potenzieren wird sinnvollerweise im allgemeinen Rahmen einer Halbgruppe H mit Einselement 1 abgehandelt. Das Problem besteht dann darin, für $x \in H$ die Potenz x^n mit einer natürlichen Zahl n, also das Produkt aus n Faktoren x durch *möglichst wenige Multiplikationen* auszudrücken. Die ganz naive Methode,

$$x^n = x \cdots x,$$

benötigt $n-1$ Multiplikationen. Der Aufwand ist proportional zu n, wächst also *exponentiell* mit der Stellenzahl von n. Die bessere Idee ist das **binäre Potenzieren**, das im Falle einer additiv geschriebenen Verknüpfung (insbesondere für die Halbgruppe $H = \mathbb{N}$) auch als russische Bauernmultiplikation bekannt ist, und schon bei den Ägyptern um 1800 v. Chr. sowie bei den Indern vor 200 v. Chr. verwendet wurde.

Die Beschreibung des Verfahrens beginnt bei der binären Darstellung des Exponenten n (o. B. d. A. $n > 0$),

$$n = b_k 2^k + \cdots + b_0 2^0 \qquad \text{mit } b_i \in \{0,1\},\ b_k = 1,$$

also $k = \lfloor {}^2\log n \rfloor = \ell(n) - 1$. Dann ist

$$x^n = (x^{2^k})^{b_k} \cdots (x^2)^{b_1} \cdot x^{b_0}.$$

Man erzeugt also der Reihe nach $x, x^2, x^4, \ldots, x^{2^k}$, indem man k-mal quadriert, und multipliziert dann die x^{2^i} mit $b_i = 1$ miteinander; das sind $\nu(n)$ Stück, wobei $\nu(n)$ die Anzahl der Einsen in der binären Darstellung ist. Insbesondere ist $\nu(n) \le \ell(n)$. Insgesamt muss man also $\ell(n) + \nu(n) - 2$ Multiplikationen ausführen.

Damit ist gezeigt:

Satz 1 *Sei H eine Halbgruppe mit 1. Dann lässt sich x^n für alle $x \in H$ und $n \in \mathbb{N}$ mit höchstens $2 \cdot \lfloor {}^2\log n \rfloor$ Multiplikationen berechnen.*

Der Aufwand ist also nur *linear* in der Stellenzahl von n. Natürlich muss man zur Abschätzung des gesamten Rechenbedarfs auch den Aufwand für die Multiplikation in der Halbgruppe H berücksichtigen.

Eine Beschreibung in Pseudocode sieht so aus:

Prozedur BinPot
Eingabeparameter:
x = zu potenzierender Wert
[dient lokal zur Speicherung der iterativen Quadrate].
n = Exponent.
Ausgabeparameter:

y = Ergebnis x^n
[dient lokal zur Akkumulation des Teilprodukts].

Anweisungen:

$y := 1$.
Solange $n > 0$:
Falls n ungerade: $y := yx$.
$x := x^2$.
$n := \lfloor n/2 \rfloor$.

Anmerkungen

1. Der Algorithmus ist im wesentlichen, aber doch nicht ganz optimal. Mit der Theorie der „Additionsketten“ aus der Zahlentheorie kann man zeigen, dass die durchschnittlich benötigte Zahl der Multiplikationen sich asymptotisch wie ${}^2\log n$ verhält, also nur halb so groß ist.

2. Die unterschiedliche Zahl von benötigten Multiplikationen je nach Exponent ist Ansatzpunkt der *Zeitbedarfs-* und der *Stromverbrauchsanalyse* (timing attacks, power attacks nach Paul Kocher), wo ein Gerät, etwa eine Chipkarte, das mit einem geheimen Exponenten potenziert, analysiert wird, um ihm das Geheimnis zu entlocken.