3 Rechnen mit ganzen Zahlen

3.1 Addition von Zahlen in b-adischer Darstellung

- Wir verstehen laut Definition unter der Laufzeit eines Algorithmus die Zahl elementarer Rechenoperationen.
- Diese Definition liegt der Annahme zugrunde, dass jede dieser elementaren Operationen in konstant vielen Schritten ausgeführt wird.
- Diese Annahme ist nur dann realistisch, wenn diese Operationen auf Zahlen beschränkter Größe angewandt werden.

Algorithm 1 Addition b-adischer Zahlen

Require: $x = (x_{k-1} \dots x_1 x_0)_b, y = (y_{k-1} \dots y_1 y_0)_b, b \geq 2$
  $c \leftarrow 0$
  for $j \leftarrow 0$ To $k-1$ **do**
    $s_j \leftarrow (x_j + y_j + c) \mod b$
    $c \leftarrow \lfloor \frac{x_j + y_j + c}{b} \rfloor$
  end for
  $s_k \leftarrow c$
  return $(s_k s_{k-1} \dots s_1 s_0)$

Satz 3.1. *Der Algorithmus ist korrekt und hat eine Laufzeitfunktion in $\mathcal{O}(k)$.*

Beweis per Induktion über k. Für jedes $k \geq 1$ arbeitet der Algorithmus korrekt; es gilt immer $c \in \{0, 1\}$.

Induktionsanfang ($k = 1$): $s_0 = (x_0 + y_0 + c) \mod b$, $s_1 = \lfloor \frac{x_0 + y_0 + c}{b} \rfloor = c$

$$(s_1 s_0)_b = \lfloor \frac{x_0 + y_0}{b} \rfloor \cdot b + (x_0 + y_0) \mod b = x_0 + y_0$$

Außerdem gilt $0 \leq x + y_0 \leq 2b - 2$ und daher $c \in \{0, 1\}$.

Induktionsschritt ($k \to k+1$): Die Behauptung gelte für ein beliebiges, aber festes $k \geq 1$. Dann ist

$$(x_{k-1} \dots x_0)_b + (y_{k-1} \dots y_0)_b = (c s_{k-1} \dots s_0)_b$$

also gilt für $x = (x_k x_{k-1} \dots x_0)$ und $y = (y_k y_{k-1} \dots y_0)$:

$$\begin{aligned}
x + y &= x_k b^k + y_k b^k + (x_{k-1} \dots x_0)_b + (y_{k-1} \dots y_0) \\
&= (x_k + y_k + c) b^k + (s_{k-1} \dots s_0)_b \\
&= (s_{k+1} s_k) b^k + (s_{k-1} \dots s_0) \\
&= (s_{k+1} s_k \dots s_0)_b
\end{aligned}$$

und $c' = \lfloor \frac{x_k + y_k + c}{b} \rfloor \in \{0, 1\}$, da $0 \leq x_k + y_k + c \leq 2b - 2 + 1 < 2b$ □

Bemerkung.
- Der Algorithmus kann mittels der allgemeinen Komplementdarstellung auch zur Addition ganzer Zahlen verwendet werden.
- Insbesondere kann man den Algorithmus auch zur Subtraktion nutzen, indem man $f(-x) = K_b^\ell(x)$ verwendet.
- Auch $f(-x)$ kann in Laufzeit von $\mathcal{O}(k)$ berechnet werden, sodass die Subtraktion ebenfalls Laufzeit von $\mathcal{O}(k)$ hat.

3.2 Multiplikation ganzer Zahlen in b-adischer Darstellung

Die naive schriftliche Multiplikation zweier k-stelliger Zahlen hat Laufzeit $\Theta(k^2)$.

$$\left(\sum_{j=0}^{k-1} x_j b^j\right)\left(\sum_{j=0}^{k-1} y_j b^j\right)$$

Beispiel. Betrachte $x = (x_1 x_0)_b$ und $y = (y_1 y_0)_b$ für $b \geq 2$ und berechne

$$x \cdot y = \underbrace{(x_1 y_1)}_{:=p} b^2 + \underbrace{(x_1 y_0 + y_1 x_0)}_{r-p-q} b + \underbrace{(x_0 y_0)}_{:=q}$$

Rechenschritte: 4 Multiplikationen und 3 Additionen.

Alternative Idee: Berechne $p := x_1 y_1, q = x_0 y_0, r = (x_1 + x_0) \cdot (y_1 + y_0)$.

$$x \cdot y = pb^2 + (r - p - q)b + q$$

Rechenschritte: 3 Multiplikationen, 6 Additionen
Es seien $x = (x_3 x_2 x_1 x_0), y = (y_3 y_2 y_1 y_0)$. Berechne

$$\begin{aligned}
p &:= (x_3 x_2)_b \cdot (y_3 y_2)_b \\
q &:= (x_1 x_0)_b \cdot (y_1 y_0)_b \\
r &:= \big((x_3 x_2)_b + (x_1 x_0)_b\big) \cdot \big((y_3 y_2)_b + (y_1 y_0)_b\big)
\end{aligned}$$

und verwende dabei die alternative Idee. Rechenschritte: 9 Multiplikationen und $3 \cdot 6 + 2$ Additionen.

Dann ist

$$x \cdot y = pb^4 + (r - p - q)b^2 + q$$

Rechenschritte gesamt: 9 Multiplikationen, 24 Additionen.

3.3 Karazubas Multiplikation

Algorithm 2 Karazubas Multiplikation

Require: $x, y \in \mathbb{N}$ in Binärdarstellung
  if $x < 8$ and $y < 8$ **then return** $x \cdot y$
  else
    $\ell \leftarrow 1 + \lfloor \log_2(\max\{x, y\}) \rfloor$
    $k \leftarrow \lfloor \frac{\ell}{2} \rfloor, b \leftarrow 2^k$
    $x' \leftarrow \lfloor \frac{x}{b} \rfloor, x'' \leftarrow x \mod b$
    $y' \leftarrow \lfloor \frac{y}{b} \rfloor, y'' \leftarrow y \mod b$
    $p \leftarrow x' \cdot y'$
    $q \leftarrow x'' \cdot y''$
    $r \leftarrow (x' + x'') \cdot (y' + y'')$ **return** $pb^2 + (r - p - q)b + q$
  end if

Beispiel. $x = (1100011)_2, y = (0010110)_2$

$\ell := 1 + \lfloor \log_2(x) \rfloor = 7, k = \lfloor \frac{\ell}{2} \rfloor, b = 2^k = 8$

$$\begin{aligned}
p &:= x' \cdot y' = (1100)_2 \cdot (0010)_2 = (11000)_2 \\
q &:= x'' \cdot y'' = (011)_2 \cdot (110)_2 = (10010)_2 \\
r &:= (x' + x'') \cdot (y' + y'') = (1111)_2 + (1000)_2 = (1111000)_2 \\
pb^2 + (r - p - q)b + q &= (11000)_2 2^6 + (1001110)_2 2^3 + (10010)_2 \\
&= (11000000000)_2 + (1001110000)_2 + (10010)_2 \\
&= (100010000010)_2
\end{aligned}$$

3.4 Karazubas Multiplikation – Korrektheit und Laufzeit

Satz 3.2. *Der Algorithmus ist korrekt und hat eine Laufzeitfunktion in* $\mathcal{O}(\ell^{\log_2(3)})$ *mit* $\ell := 1 + \lfloor \log_2(\max\{x, y\}) \rfloor$

Bemerkung.
- $\log_2(3) < 1.59; \ell = 1\,000 \Longrightarrow \ell^2 = 1\,000\,000, \ell^{\log_2(3)} < 60\,000$
- Es gibt schnellere Verfahren (z. B. Schönhage-Strassen-Algorithmus)
- Ganzzahlige Division kann auf Multiplikation zurückgeführt werden.

Laufzeitanalyse. Es sei $T(\ell)$ maximale Laufzeit für Imputzahlen mit $\leq \ell$ Ziffern.

$$T(\ell) \leq \begin{cases} c & \text{für } \ell \leq 3, \\ c \cdot \ell + 3T\left(\left\lceil \frac{\ell}{2} \right\rceil\right) + 1 & \text{für } \ell \geq 4 \end{cases}$$

$\square$