LANGZAHLARITHMETIK

1. Motivation

(Beschränkungen heutiger Computer): Ein 32-Bit Rechner kann lediglich mit ganzen Zahlen rechnen, die eine Länge von 32 Bit nicht übersteigen. Hierbei gibt es jedoch Probleme, da ja bei der Addition zweier 32-Bit Zahlen, das Ergebnis nicht mehr zwangsläufig mit 32 Bits darstellbar sein muss. Bei der Multiplikation zweier 32-Bit Zahlen ist die Situation noch etwas schlimmer, da das Ergebnis bis zu 64 Bits beanspruchen kann. Exakte Ergebnisse kann man also nur erwarten, wenn man die Länge der Operanden auf 16 Bits begrenzt. Mit dieser Begrenzung will man sich aber nicht zufrieden geben, d.h. man sucht nach einer Möglichkeit mit "beliebig" langen Zahlen zu rechnen.

Hierzu wurden zwei Klassen entworfen, die Klasse LongInt und die Klasse Rational. LongInt dient zur Repräsentation von ganzen Zahlen und Rational zur Repräsentation von rationalen Zahlen.

2. Beschreibung der Klasse LongInt

2.1. **Aufbau der Klasse.** Die Klasse LongInt ist abgeleitet von einem Vector (eine Container-Klasse der STL). Durch Ableitung der Klasse LongInt von einem Vector, vererbt man sämtliche Funktionalität, die der Vector besitzt an die Klasse LongInt, d.h. LongInt wird damit quasi selbst zu einem Vector, nur mit dem Unterschied, dass man der Klasse LongInt noch weitere Funktionen hinzufügen kann.

Bemerkung. (Notation)
Im folgenden wird desöfteren die Rede sein von:
$u = (u_{n-1}|u_{n-2}|...|u_0)_\beta$ (wobei u_i, β ganze Zahlen sind, mit $0 \leq u_i < \beta$, $\beta > 0$).
Dies soll einfach heissen, dass $u = \sum_{i=0}^{n-1} u_i \beta^i$ ist.

2.2. **Abspeicherung von ganzen Zahlen.** Nun stellt sich die Frage: Wie kann man denn nun "beliebig" lange ganze Zahlen abspeichern bzw. repräsentieren?
Die Antwort lautet: Indem man sie im Zehntausendersystem darstellt und die Stellen, die sich bzgl. dieser Darstellung ergeben sukzessive in den Vector „hineinschiebt".

Beispiel. (Abspeicherung)
ausgelassen

D.h. man zerteilt die gegebene Dezimalzahl, von rechts beginnend, in Stücke der Länge 4. Die Funktion, die dies bewerkstelligt ist: operator=(char const * const src). Übergabeparameter an die Funktion ist die Dezimalzahl in Form eines nullterminierten Strings.

Bemerkung. Ich habe mich für eine Darstellung im Zehntausendersystem entschieden, weil das Ganze damit, meiner Meinung nach, etwas anschaulicher wird (man könnte aber auch jede andere beliebige Basis β nehmen – Voraussetzung ist aber, dass sich $\beta - 1$ mit maximal 16 Bits darstellen laesst).

Nun stellt sich als nächstes die Frage: Wie kann man mit diesen LongInts rechnen, d.h. wie implementiert man die elementaren Operationen (+, -, *, /)?

2.3. **Addition.**

Beispiel. (Addition)
ausgelassen

Allgemein: Seien u, v pos. Zahlen der β-Länge n bzw. m, $n \geq m$
d.h $u = (u_{n-1}|u_{n-2}|...|u_0)_\beta$ und $v = (v_{m-1}|v_{m-2}|...|v_0)_\beta$, dann gewinnt man das Ergebnis $w = u + v$ folgendermassen:

Schritt1: Setze $c = 0$.
Schritt2: Für alle $i = 0$ bis $m - 1$,
setze: $w_i = u_i + v_i + c$.
Falls $w_i \geq \beta$, setze $w_i = w_i - \beta$ und $c = 1$,
ansonsten setze $c = 0$.
Schritt3: Für alle $i = m$ bis $n - 1$,
setze: $w_i = u_i + c$.
Falls $w_i \geq \beta$, setze $w_i = w_i - \beta$ und $c = 1$,
ansonsten setze $c = 0$.
Schritt4: Falls $c = 1$, setze $w_n = c$.

2.4. **Subtraktion.**

Beispiel. (Subtraktion)
ausgelassen

Allgemein: Seien u, v wieder pos. Zahlen der β-Länge n bzw. m, $u \geq v$
d.h $u = (u_{n-1}|u_{n-2}|...|u_0)_\beta$ und $v = (v_{m-1}|v_{m-2}|...|v_0)_\beta$, dann gewinnt man das Ergebnis $w = u - v$ folgendermassen:

Schritt1: Setze $c = 0$.
Schritt2: Für alle $i = 0$ bis $m - 1$,
setze: $w_i = u_i - v_i - c$.
Falls $w_i < 0$, setze $w_i = w_i + \beta$ und $c = 1$,
ansonsten setze $c = 0$.
Schritt3: Für alle $i = m$ bis $n - 1$,
setze: $w_i = u_i - c$.
Falls $w_i < 0$, setze $w_i = w_i + \beta$ und $c = 1$,
ansonsten setze $c = 0$.

2.5. **Multiplikation.**

2.5.1. *Schulmethode.*

Beispiel. (Schulmultiplikation)
ausgelassen

Allgemein: Seien u, v pos. Zahlen der β-Länge n bzw. m,
d.h $u = (u_{n-1}|u_{n-2}|...|u_0)_\beta$ und $v = (v_{m-1}|v_{m-2}|...|v_0)_\beta$, dann gewinnt man das Ergebnis $w = u * v$ folgendermassen:

Schritt1: Setze $w = 0$.
Schritt2: Für alle $i = 0$ bis $m - 1$, und
für alle $j = 0$ bis $n - 1$,
setze: $res = u_j * v_i$
Splitte res in die höherwertige (hi)
und niedrigwertige (lo) Stelle auf.
Setze $tmp = (hi|lo|\underbrace{0|...|0}_{}{}_\beta$.
Setze $w = w + tmp$.

Die Rechenzeit zur Multiplikation zweier Zahlen der Länge n (nach der Schulmethode) beträgt: $\mathcal{O}(n^2)$.

2.5.2. *Karatsuba-Methode.* Es gibt aber noch eine andere Multiplikationsmethode, die sog. Karatsuba-Multiplikation, die eine geringere Rechenzeit von $\mathcal{O}(n^{\log_2(3)})$ benötigt (diese wird aber erst erreicht, wenn die Zahlen eine gewisse Länge übersteigen).

Bemerkung. (Karatsuba) Anatolii Alexeevich Karatsuba war ein russischer Mathematiker, der dieses Multiplikationsverfahren im Jahre 1962 entwickelt hat.

Diese Multiplikation funktioniert nach dem folgendem Prinzip:

Seien u, v pos. Zahlen der β-Länge n (o.B.d.A. sei n gerade, ansonsten erweitere u, v mit führenden Nullen), d.h $u = (u_{n-1}|u_{n-2}|...|u_0)_\beta$ und $v = (v_{n-1}|v_{n-2}|...|v_0)_\beta$, dann lassen sich u und v wie folgt schreiben:

$u = a * \beta^{\frac{n}{2}} + b$ und $v = c * \beta^{\frac{n}{2}} + d$, wobei a, b, c, d Zahlen der β-Länge $\frac{n}{2}$ sind.
Somit ergibt sich für das Produkt $u * v$:

$$\begin{aligned} u * v &= (a\beta^{\frac{n}{2}} + b) * (c\beta^{\frac{n}{2}} + d) \\ &= \beta^n ac + \beta^{\frac{n}{2}}[ad + bc] + bd \end{aligned}$$

Auf den ersten Blick hat man nicht viel gewonnen, man hat eine Multiplikation zweier Zahlen der Länge n durch 4 Multiplikationen mit Zahlen der Länge $\frac{n}{2}$ ersetzt. Der Aufwand ist somit derselbe, da ja $4 * (\frac{n}{2})^2 = n^2$. Man macht sich nun jedoch zunutze, dass $ad + bc = ac + bd + (a - b) * (d - c)$ ist. Man erhält damit:

$$u * v = \beta^n ac + \beta^{\frac{n}{2}}[ac + bd + (a - b)(d - c)] + bd$$

D.h. man kann die Multiplikation zweier n-stelliger Zahlen zurückführen auf 3 Multiplikationen von $\frac{n}{2}$-stelligen Zahlen und einigen Additionen. Für den Aufwand $T(n)$ zur Multiplikation zweier n-stelliger Zahlen gilt damit folgende Rekursionsformel: $T(n) = 3 * T(\frac{n}{2}) + n$. Setzt man $T(1) = 1$, so erhält man daraus:

$$T(n) = 3n^{\log_2(3)} - 2n \ (*)$$

Dies entspricht einer Rechenzeit von $\mathcal{O}(n^{\log_2(3)})$.

Beweis. (zu *) per Induktion:
ausgelassen □

Beispiel. (Karatsuba)
ausgelassen

2.6. **Division.**

Beispiel. (Division)
ausgelassen

Der erste Schritt besteht darin, für zwei nichtnegative Zahlen u, v der Form $u = (u_m|u_{m-1}|...|u_0)_\beta$, $v = (v_{m-1}|v_{m-2}|...|v_0)_\beta$ mit $\frac{u}{v} < \beta$ eine Zahl q zu finden, so dass $q = \lfloor \frac{u}{v} \rfloor$. Insbesondere ist q dann die eindeutige bestimmte Zahl, die $0 \leq u - qv < v$ erfüllt. Wie bestimmen wir also dieses q ?

Ein Versuch basiert darauf, dass man die führenden Stellen u_m, u_{m-1} von u und v_{m-1} von v betrachtet und definiert:

$$\hat{q} := \min(\left\lfloor \frac{u_m\beta + u_{m-1}}{v_{m-1}} \right\rfloor, \beta - 1)$$

Es wird sich herausstellen, dass dieses $\hat{q}$ eine sehr gute Abschätzung für den wahren Wert q liefert, wenn v_{m-1} groß genug ist. Um zu prüfen, wie nahe $\hat{q}$ dem gewünschtem q kommt, wird nun zuerst beweisen, dass $\hat{q}$ niemals zu klein ist.

Theorem. *Unter der Notation von eben gilt:* $\hat{q} \geq q$.

Beweis. Da $q \leq \beta - 1$ ist die Behauptung sicherlich wahr, wenn $\hat{q} = \beta - 1$.

Andernfalls gilt: $\hat{q} = \left\lfloor \frac{u_m\beta + u_{m-1}}{v_{m-1}} \right\rfloor$ Also: $\hat{q} > \frac{u_m\beta + u_{m-1}}{v_{m-1}} - 1$

$\Longrightarrow \hat{q}v_{m-1} > u_m\beta + u_{m-1} - v_{m-1} \Longrightarrow \hat{q}v_{m-1} \geq u_m\beta + u_{m-1} - v_{m-1} + 1$

Es folgt, dass:

$$\begin{aligned} u - \hat{q}v &\leq u - \hat{q}v_{m-1}\beta^{m-1} \\ &\leq u_m\beta^m + u_{m-1}\beta^{m-1} + ... + u_1\beta + u_0 - (u_m\beta^m + u_{m-1}\beta^{m-1} - v_{m-1}\beta^{m-1} + \beta^{m-1}) \\ &= \underbrace{u_{m-2}\beta^{m-2} + ... + u_1\beta + u_0 - \beta^{m-1}}_{} + v_{m-1}\beta^{m-1} < v_{m-1}\beta^{m-1} \leq v \end{aligned}$$

Es gilt also: $u - \hat{q}v < v$.

Da q die eindeutig bestimmte Zahl mit $0 \leq u - qv < v$ ist $\Longrightarrow \hat{q} \geq q$ □

Theorem. *Wenn* $v_{m-1} \geq \left\lfloor \frac{\beta}{2} \right\rfloor$, *dann ist* $\hat{q} - 2 \leq q \leq \hat{q}$.

Beweis. Angenommen $\hat{q} \geq q + 3$. Dann haben wir:

$$\begin{aligned} \hat{q} &\leq \frac{u_m\beta + u_{m-1}}{v_{m-1}} = \frac{u_m\beta^m + u_{m-1}\beta^{m-1}}{v_{m-1}\beta^{m-1}} \\ &\leq \frac{u}{v_{m-1}\beta^{m-1}} < \frac{u}{v - \beta^{m-1}}, \text{ (weil } v < (v_{m-1} + 1)\beta^{m-1}) \end{aligned}$$

(Der Fall $v = \beta^{m-1}$ ist nicht möglich, weil sonst wäre $v = (1|0|...|0)_\beta$,

$\Longrightarrow \frac{u_m\beta + u_{m-1}}{v_{m-1}} = \frac{u_m\beta^m + u_{m-1}\beta^{m-1}}{v_{m-1}\beta^{m-1}} = \frac{u_m\beta^m + u_{m-1}\beta^{m-1}}{v} \leq \frac{u}{v} < \beta$,

folglich wäre dann $u - \hat{q}v \geq u - u_m\beta^m - u_{m-1}\beta^{m-1} \geq 0$.

$\Longrightarrow q = \hat{q}$, im Widerspruch zur Annahme)

Weiter gilt: $q = \lfloor \frac{u}{v} \rfloor \Longrightarrow q > \frac{u}{v} - 1$

Und somit: $3 \leq \hat{q} - q < \frac{u}{v - \beta^{m-1}} - \frac{u}{v} + 1 = \frac{u}{v}(\frac{\beta^{m-1}}{v - \beta^{m-1}}) + 1$

$$\Longrightarrow \frac{u}{v} > 2(\frac{v - \beta^{m-1}}{\beta^{m-1}}) \geq 2(v_{m-1} - 1)$$

Da $\beta - 4 \geq \hat{q} - 3 \geq q = \lfloor \frac{u}{v} \rfloor \geq 2(v_{m-1} - 1)$

$\Longrightarrow v_{m-1} < \left\lfloor \frac{\beta}{2} \right\rfloor$, (im Widerspruch zur Voraussetzung) □

Theorem. *Sei* $\beta^{m-1} \leq v < \beta^m$.

Definiere $u' := u * \left\lfloor \frac{\beta}{v_{m-1}+1} \right\rfloor$ *und* $v' := v * \left\lfloor \frac{\beta}{v_{m-1}+1} \right\rfloor$.

Dann gelten:

(1) $\frac{u'}{v'} = \frac{u}{v}$
(2) $\beta^{m-1} \leq v' < \beta^m$
(3) $v'_{m-1} \geq \left\lfloor \frac{\beta}{2} \right\rfloor$

Beweis. zu (1): ist offensichtlich

zu (2):

$$\begin{aligned} \beta^{m-1} &\leq v \leq v * \underbrace{\left\lfloor \frac{\beta}{v_{m-1} + 1} \right\rfloor} < (v_{m-1} + 1) * \beta^{m-1} * \left\lfloor \frac{\beta}{v_{m-1} + 1} \right\rfloor \\ &\leq (v_{m-1} + 1) * \beta^{m-1} * (\frac{\beta}{v_{m-1} + 1}) \leq \beta^m \end{aligned}$$

$\Longrightarrow \beta^{m-1} \leq v' < \beta^m$

zu (3): Wenn $v_{m-1} \geq \left\lfloor \frac{\beta}{2} \right\rfloor$, dann gilt: $v_{m-1} * \underbrace{\left\lfloor \frac{\beta}{v_{m-1}+1} \right\rfloor} \geq v_{m-1} \geq \left\lfloor \frac{\beta}{2} \right\rfloor$

$\Longrightarrow v'_{m-1} \geq \left\lfloor \frac{\beta}{2} \right\rfloor$.

Andernfalls, also wenn $1 \leq v_{m-1} < \left\lfloor \frac{\beta}{2} \right\rfloor$ ($\Longrightarrow$ (**) $1 \leq v_{m-1} \leq \frac{\beta}{2} - 1$)

$\Longrightarrow v_{m-1} * \left\lfloor \frac{\beta}{v_{m-1}+1} \right\rfloor > v_{m-1} * (\frac{\beta}{v_{m-1}+1} - 1) \geq \frac{\beta}{2} - 1 \geq \left\lfloor \frac{\beta}{2} \right\rfloor - 1$

$\Longrightarrow v'_{m-1} \geq \left\lfloor \frac{\beta}{2} \right\rfloor$

(zu (***): $v_{m-1} * (\frac{\beta}{v_{m-1}+1} - 1) - (\frac{\beta}{2} - 1)$

$= \frac{\beta v_{m-1} + (-v_{m-1} - \frac{\beta}{2} + 1)(v_{m-1}+1)}{v_{m-1}+1} = \frac{(\frac{\beta}{2} - v_{m-1} - 1)(v_{m-1}-1)}{v_{m-1}+1} \geq 0,$

wegen (**) und Voraussetzung) □

Damit sind jetzt sämtliche Vorkehrungen getroffen, um einen geeigneten Divisionsalgorithmus zu formulieren:

Seien $U = (U_{n-1}|U_{n-2}|...|U_0)_\beta$, $v = (v_{m-1}|v_{m-2}|...|v_0)_\beta$
nicht-negative Zahlen mit: $U \geq v$ und $v_{m-1} \neq 0$.
Dann lassen sich $q = \lfloor \frac{U}{v} \rfloor$ und $r = U - qv$ wie folgt berechnen.

`Schritt1:` `Setze` $d = \left\lfloor \frac{\beta}{v_{m-1}+1} \right\rfloor$,
$U' = U * d = (U'_{k-1}|U'_{k-2}|...|U_0)$ `und`
$v' = v * d = (v'_{m-1}|v'_{m-2}|...|v_0)$.

`Schritt2:` `Falls` $\frac{U'_{k-1}\beta + U'_{k-2}}{v'_{m-1}} \geq \beta$,
`erweitere` U' `mit führender Null und`
`setze` $k = k + 1$.

`Schritt3:` `Setze` $i = 0$

`Schritt4:` `Setze`
$(u'_m|u'_{m-1}|...|u'_0) = (U'_{k-i-1}|U'_{k-i-2}|...|U'_{k-i-m-1})$

`Schritt5:` `Setze` $\hat{q} = \min(\left\lfloor \frac{u'_m\beta + u'_{m-1}}{v'_{m-1}} \right\rfloor, \beta - 1)$

`Schritt6:` `Setze` $u' = u' - \hat{q}v$

`Schritt7:` `Falls` $u' < 0$, `setze` $u' = u' + v$, $\hat{q} = \hat{q} - 1$

`Schritt8:` `Falls` $u' < 0$, `setze` $u' = u' + v$, $\hat{q} = \hat{q} - 1$

`Schritt9:` `Setze`
$(U'_{k-i-1}|U'_{k-i-2}|...|U'_{k-i-m-1}) = (u'_m|u'_{m-1}|...|u'_0)$,
$q_{k-m-i-1} = \hat{q}$, $i = i + 1$

`Schritt10:` `Falls` $i \leq k - m - 1$ `gehe wieder zu Schritt4,`
`ansonsten gehe zu Schritt11.`

`Schritt11:` `Setze` $r = \left\lfloor \frac{U'}{d} \right\rfloor$.

3. Beschreibung der Klasse Rational

3.1. **Aufbau der Klasse.** Die Klasse besitzt zwei Elemente vom Datentyp LongInt, einen Zähler und einen Nenner.
Desweiteren besitzt sie wiederum Funktionen zum Addieren, Subtrahieren, Multiplizieren und Dividieren, sowie einige Hilfsfunktion, etwa zum Kuerzen oder zur Ermittlung des Hauptnenners.

3.2. **Abspeicherung von rationalen Zahlen.** Sei eine rationale Zahl in Form eines nullterminierten Strings gegeben, wobei Zähler und Nenner innerhalb dieses Strings durch ein „/" voneinander abgetrennt sind.

Dieser String wird in der Funktion: operator=(const char * const src) in seine beiden Bestandteile zerlegt und zaehler bzw. nenner werden damit entprechend initialisiert.

3.3. **Kurze Beschreibung der Funktionen.** Saemtliche Funktionen dieser Klasse werden auf entprechende Funktionen der Klasse LongInt zurückgeführt, weshalb auf die Beschreibung dieser Funktionen verzichtet wird.

4. Fazit

Nun sind wir in der Lage (fast) alle Algorithmen, die wir in diesem Seminar kennengelernt haben über dem Körper $\mathbb{Q}$ durchzuführen. Hierzu muss lediglich in der Headerdatei compalg1.h zu anfangs ein *#include „rational.h"* eingefügt werden und *typedef Fp<5> K* muss durch *typedef Rational K* ersetzt werden.