Bemerkungen zur ersten Übung

1 ScanLine-Algorithmus

In der Vorlesung wurde ein Algorithmus gesucht, der die folgende Aufgabenstellung löst: *Bestimme aus einer gegebenen endlichen Folge von N reellen Zahlen diejenige Teilfolge, deren Summe aller Folgenelemente maximal ist. Wie groß ist diese Summe?* Durch stupides Ausprobieren erhält man die Laufzeit $O(N^3)$, durch ein Devide–And–Conquer–Verfahren kann man diese auf $O(N \log N)$ verbessern. Es gibt jedoch auch ein Verfahren, das in $O(N)$ läuft:

PROCEDURE ScanLine (X, N)
    BEGIN
    $ScanMax := 0$
    $BisMax := 0$
    FOR $i := 1$ **TO** N
        BEGIN
        $a := X[i]$
        IF $ScanMax + a > 0$ **THEN**
            $ScanMax := ScanMax + a$
        ELSE
            $ScanMax := 0$
        IF $ScanMax > BisMax$ **THEN**
            BisMax:=ScanMax
        END
    RETURN $BisMax$

1.1 Funktionsweise

Es sind folgende Beobachtungen zu machen:

- Die Folge wird von links nach rechts bearbeitet, da nur *zusammenhängende* Teilfolgen gesucht werden.
- Da nur die Summe interressiert, werden keine Indizes auf den Anfang oder das Ende der besten Teilfolge gespeichert.
- Die Suche nach der besten Teilfolge geschieht dadurch, daß man versucht, sukzessiv die aktuelle Teilfolge (deren Summe in $ScanMax$ gespeichert ist) rechts durch ein weiteres Element a zu ergänzen. Dabei gilt folgendes:
  - Hat die aktuelle Teilfolge, die durch Anhängen des neuen Elementes entsteht, eine *negative* Summe, so kann sie komplett vernachlässigt werden. Man fängt mit einer neuen aktuellen Teilfolge an.
  - Hat die neue Teilfolge jedoch eine positive Summe, so kann im nächsten Durchlauf nochmals erweitert werden.
- Nach jeder Erweiterung der aktuellen Teilfolge wird überprüft, ob das bisherige Maximum $BisMax$ überschritten wurde.

1.2 Laufzeit

Die Laufzeit ist sofort klar, wenn man bedenkt, daß die Schleife genau N mal durchlaufen wird und innerhalb der Schleife nur Operationen stehen, die in $O(1)$ bearbeitet werden können.

2 Ein Multiplikationsverfahren

Gegeben sei der folgende Multiplikationsalgorithmus, der im Prinzip das in der Schule benutzte Verfahren zur Multiplikation zweier Zahlen a und b nachvollzieht. Der Einfachheit wird angenommen, daß beide Zahlen positiv, ganzzahlig und binär sind.

```
PROCEDURE Multiplikation (a, b, z)
    BEGIN
    x := a
    y := b
    z := 0
    WHILE y > 0
        BEGIN
        IF odd(y) THEN
            BEGIN
            y := y − 1
            z := z + x
            END
        ELSE
            BEGIN
            y := y/2
            x := x * 2
            END
        END
    RETURN  z
    END
```

2.1 Funktionsweise

In x steht der Multiplikant, in y der Multiplikator. In jedem Durchlauf wird überprüft, ob $odd(y)$ wahr ist, d.h. ob das unterste Bit (LSB) von y gesetzt ist. Wenn ja, muß der aktuelle Multiplikant zum Ergebnis addiert werden. Außerdem wird y um eins erniedrigt, damit das unterste Bit von y gelöscht wird und im nächsten Durchlauf der andere Teil der *IF*–Anweisung durchlaufen wird. Wenn das unterste Bit jedoch nicht gesetzt ist, wird der Multiplikant eine Stelle nach links verschoben (durch Multiplikation mit 2, denn wir rechnen im Dualsystem) und der Multiplikator um eine Stelle nach rechts (durch Division durch 2, s.o.). Die Schleife wird solange durchlaufen, bis der Multiplikator $y = 0$ ist.

2.2 Beweis der Korrektheit

Für die Korrektheit des Algorithmus sind zwei Dinge zu zeigen:

1. Der Algorithmus bricht für jede beliebige Kombination der beiden Input-Werte a, b nach endlich vielen Schritten ab.

2. Das Produkt ist korrekt berechnet worden.

2.2.1 zu 1)

Man sieht, daß in jedem Schleifendurchlauf entweder y um 1 vermindert oder durch 2 geteilt wird. Die Division durch 2 ist aber im Binärsystem (solange nur mit ganzen Zahlen hantiert wird) gleichbedeutend damit, daß y um *mindestens* 1 vermindert wird. Für $y > 1$ ist das klar und durch Abrunden gilt für $y = 1$: $(1/2)_{bin} = 0$. Demnach wird y in *jedem* Durchlauf um 1 verringert und die Schleife bricht wegen der Bedingung der *WHILE*–Schleife nach höchstens y Durchläufen ab.

2.2.2 zu 2)

Für die Korrektheit des Produktes betrachtet man folgende *Schleifeninvariante*:

$$P(x, y, z, a, b) : y \geq 0 \quad \text{und} \quad z + xy = ab$$

Man muß nun zeigen, daß P

- *vor* der Schleife (Induktionsverankerung) und
- *während* der Schleife (Induktionschritt)

erfüllt ist. Vor der Schleife ist das durch die Initialisierung von a, b und z trivial. Während eines Durchlaufs nimmt an, daß die Invariante am Anfang der Schleife korrekt ist und schließt daraus, daß sie auch am Ende der Schleife ebenfalls erfüllt wird (Nachrechnen!) und zwar unabhängig von der Verzweigung der *IF*–Anweisung.

Nach dem Beenden der Schleife (was wegen Punkt 1 auch wirklich eintritt) hat man folgende Konstellation: Nach der Invarianten muß $y \geq 0$ sein, nach der Schleifenbedingung, die nach Abbruch der Schleife nicht mehr erfüllt ist, muß $y \leq 0$ sein. Demnach ist $y = 0$. Eingesetzt ist die Invariante folgt daraus $z + 0 = z = ab$. Also ist z das gesuchte Produkt.

2.3 Laufzeit

Oben wurde gesagt, daß die *WHILE*–Schleife höchstens y mal durchlaufen wird. Das ist jedoch eine viel zu grobe Abschätzung. Man sieht leicht, daß y spätestens jeden zweiten Durchlauf halbiert wird. Sei nun $length(b)$ die Anzahl der Binärstellenvon b. Da y nur höchstens so oft halbiert werden kann, wie es Stellen hat, ergibt sich als Laufzeit $O(length(b))$, wenn man zusätzlich annimmt, daß *alle* arithmetischen Operationen in konstanter Zeit $O(1)$ durchgeführt werden können. Realistischer ist es jedoch, für die Addition zweier Binärzahlen a und b die Laufzeit $O(length(a) + length(b))$ anzunehmen. Dann ergibt sich

$$O(length(b) \cdot (length(a) + length(b))).$$

3 Multiplikation zweier Polynome

Es seien zwei Polynome p und q vom Grade $N - 1$ gegeben in der Form:

$$p := \sum_{j=0}^{N-1} a_j x^j \quad q := \sum_{j=0}^{N-1} b_j x^j.$$

Die Aufgabe besteht darin, die beiden Polynome miteinander zu multiplizieren und zwar derart, daß die Gesamtanzahl von Koeffizienten–Multiplikationen möglichst gering ist. Der einfachste Ansatz besteht in dem durch die mathematische Definition der Polynommultiplikation gegegebenen Algorithmus:

```
PROCEDURE PolMult (p1, p2, r)
    BEGIN
    FOR i := 0 TO 2N - 1
        r_i := 0
    FOR i := 0 TO N - 1
        FOR j := 0 TO N - 1
            r_{i+j} := r_{i+j} + a_i * b_j
    RETURN r
    END
```

Mit diesem Ansatz erhält man die Laufzeit $O(N^2)$, was man leicht an den zwei ineinander verschachtelten *IF*–Schleifen sieht. Mithilfe eines Devide–And–Conquer–Verfahrens soll nun die Laufzeit verbessert werden.

3.1 Funktionsweise

Der Einfachheit halber haben die Polynome jeweils $N = 2^k$ Koeffizienten. Die Polynome werden wie folgt aufgeteilt:

$$p(x) = p_l(x) + x^{\frac{N}{2}} p_r(x) \quad \text{mit} \quad p_l := \sum_{j=0}^{\frac{N}{2}-1} a_j x^j \quad \text{und} \quad p_r := \sum_{j=\frac{N}{2}}^{N-1} a_j x^{j-\frac{N}{2}}$$

Demnach enthält p_l die $N/2$ niedriegsten und p_r die $N/2$ höchsten Koeffizienten von p. Das Polynom q wird entsprechend geteilt. Anschließend werden die folgenden Berechnungen durchgeführt:

$$\begin{aligned} z_l(x) &:= p_l(x)q_l(x) \\ z_r(x) &:= p_r(x)q_r(x) \\ z_m(x) &:= (p_l(x) + p_r(x))(q_l(x) + q_r(x)) \end{aligned}$$

In jeder Zeile wird eine Polynommultiplikation von Polynomen der Länge $N/2$ durchgeführt. Das Ergebnispolynom wird wie folgt erhalten:

$$r(x) = p(x)q(x) = z_l(x) + (z_m(x) - z_l(x) - z_r(x))x^{\frac{N}{2}} + z_r(x)x^N \tag{1}$$

Wir haben also die Multiplation zweier Polynome der Länge N auf drei Multiplikationen von Polynomen der Länge $N/2$ zurückgeführt.

3.2 Laufzeit

Sei nun $M(N)$ die Anzahl der Koeffizienten–Multiplikationen, die für die Multiplikation zweier Polynome der Länge N notwendig sind. Dann gilt nach den obigen Überlegungen:

$$M(N) = M\left(\frac{M}{2}\right) + M\left(\frac{M}{2}\right) + M\left(\frac{M}{2}\right)$$

Die Multiplation von Polynomen der Länge $N/2$ kann natürlich nach der gleichen Vorschrift wieder zerlegt werden:

$$M(N/2) = M\left(\frac{M}{4}\right) + M\left(\frac{M}{4}\right) + M\left(\frac{M}{4}\right)$$

Diese Rekursion geht solange, bis mit $N = 1$ die Verankerung erreicht wird. Für dieses Problem wissen wir, daß wir nur eine Koeffizienten–Multiplikation durchführen müssen: $M(1) = 1$. Man erhält insgesamt:

$$\begin{aligned} M(N) &= 3^1 M\left(\frac{N}{2^1}\right) \\ &= 3^2 M\left(\frac{N}{2^2}\right) \\ \ldots \\ &= 3^k M\left(\frac{N}{2^k}\right) = 3^k M(1) = 3^k \end{aligned}$$

Dieses Ergebnis können wir jetzt umformen:

$$M(N) = 3^k = 2^{(\log 3)k} = \left(2^k\right)^{\log 3} = N^{\log 3} = N^{1.58\ldots}$$

Der oben angegebene Devide–And–Conquer–Algorithmus hat demnach eine Laufzeit von $O(N^{1.58..})$.

Warnung

Die oben beschriebene Beweisskizze für die Laufzeit ist nur deswegen so einfach, weil der Zeitaufwand für das Zusammenfügen des Gesamtergebnisses (1) als konstant $O(1)$ angesehen wurde. Im allgemeinen ist das nicht unbedingt gegeben!

4 Muster für Aufgabe 3

Als Beispiel soll die Aufgabe 3c für die Inklusion $\in$ besprochen werden. Jede Funktion $c(n)$ der linken Seite hat die Form:

$$c(n) = a(n) + b(n)$$

wobei $a(n)$ und $b(n)$ nach Vorlesung wie folgt definiert sind: es gibt Konstanten m_0 und n_0 sowie B und C derart, daß gilt:

$$\begin{aligned} |a(n)| &\leq B|f(n)| \quad \text{für} \quad n \geq m_0 \\ |b(n)| &\leq C|g(n)| \quad \text{für} \quad n \geq n_0 \end{aligned}$$

Also gilt für alle $n \geq \max(m_0, n_0)$:

$$\begin{aligned} |c(n)| &= |a(n) + b(n)| \\ &\leq |a(n)| + |b(n)| \\ &\leq B|f(n)| + C|g(n)| \\ &\leq \max(B, C)(|f(n)| + |g(n)|) \end{aligned}$$

Also ist $c(n)$ auch Element von $O(f(n) + g(n))$.