Technische Universität München
Fakultät für Informatik, I-14
Prof. Harald Räcke
Robin Münk

SoSe 2025
Übungsblatt 4
16. Mai 2025

Algorithmen und Datenstrukturen

Aufgabe 1 Rekursion

Gegeben sind die folgenden zwei Funktionen in Pseudocode.

```
func1(n):
    i = 1
    while (i <= n)
        print(i)
        i += 1
```

```
func2(n):
    i = n
    while (i > 0)
        print(i)
        i -= 1
```

Schreiben Sie beide Funktionen so um, dass keine Schleife (Wiederholung) mehr verwendet wird!

Aufgabe 2 Komplexitätsschätzung

Schätzen Sie die Komplexitäten der folgenden Algorithmen asymptotisch ab!

a) **rot13**(*string*):

```
encrypted = string;
for i = 0 to size(string)−1 {
    if (string[i] ≥ 'a' ∧ string[i] ≤ 'z') {
        abstand = string[i] − 'a';
        rotierter_abstand = (abstand + 13) mod 26;
        encrypted[i] = 'a' + rotierter_abstand;
    }
}

return encrypted;
```

b) **MatrixMultiplikation**(A, B):

```
if (cols(A)≠rows(B)) {
    return −∞;
}
n = cols(A);
C = matrix(rows(A), cols(B));
for i = 0 to rows(A)−1 {
    for j = 0 to cols(B)−1 {
        C[i][j] = 0;
        for k = 0 to n − 1 {
            C[i][j] = C[i][j] + A[i][k] · B[k][j];
        }
    }
}
return C;
```

Aufgabe 3 Komplexität der Polynom-Auswertung

In allgemeiner Form ist ein *Polynom* gegeben durch

$$p(x) := \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0 \ ,$$

wobei $a_i \in \mathbb{R}$ der Koeffizient für die i-te Potenz x^i ist. Die höchste Potenz n ist dabei der *Grad des Polynoms*. Einen einzigen Summanden $a_i x^i$ bezeichnet man als *Monom*.

Wir untersuchen nun drei verschiedene Methoden, ein Polynom auszuwerten. Gegeben sei dazu ein Array a der Länge $n+1$, wobei $a[i]$ den Koeffizienten der Potenz x^i enthält. Die allgemeine Signatur der Funktion in Pseudo-Code ist also wie folgt:

Input: Koeffizienten-Array a der Länge $n+1$, Auswertungspunkt x
Output: Ergebnis $p = a[n] \cdot x^n + ... + a[1] \cdot x + a[0]$
EvaluatePolynomial(a,x)**:**

```
p = ...
return p;
```

a) In einer ersten Implementation gehen wir vor, wie dies auf dem Papier erfolgen würde. Wir beginnen also bei der höchsten Potenz, berechnen jeweils die Monome und addieren diese dann auf:

```
p = 0;
for i = n down to 0 {
    // Berechne i-te Potenz
    m = 1;
    for j = 1 to i {
        m = m · x;
    }

    // Berechne i-tes Monom durch Multiplikation der Potenz mit dem Koeffizienten
    m = a[i] · m;

    // Aktualisiere das Polynom durch Addition des Monoms
    p = p + m;
}
```

Bestimmen Sie die Laufzeitfunktion dieser Implementierung! In welcher Effizienzklasse in O-Notation ist dieser Code?

b) Offensichtlich ist in diesem Code die Berechnung der Potenzen redundant. Wir stellen den Code nun so um, dass die Potenzen inkrementell auf Basis vorheriger Iterationen errechnet werden:

```
p = 0;
h = 1;
for i = 0 to n {
    // Berechne i-tes Monom durch Multiplikation der (vorberechneten) Potenz mit
    // dem Koeffizienten
    m = a[i] · h;

    // Aktualisiere die Potenz für den nächsten Schritt i + 1
    h = h · x;
```

```
        // Aktualisiere das Polynom durch Addition des Monoms
        p = p + m;
    }
```

Bestimmen Sie auch die Laufzeitfunktion dieser neuen Implementierung, sowohl exakt als auch in O-Notation!

c) Zuletzt untersuchen wir das sogenannte *Horner-Schema*. Die Idee ist, das Polynom umzuformen:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0 = (((\ldots) \cdot x + a_2) \cdot x + a_1) \cdot x + a_0$$

Dies kommt dadurch zustande, dass man immer wieder jeweils x aus den höheren Potenzen ausklammert. In Pseudo-Code lautet eine entsprechende Implementierung dann wie folgt:

```
p = a[n];
for i = n − 1 down to 0 {
        // Multipliziere ausgeklammertes x an den bereits ausgewerteten Teil
        p = p · x;

        // Addiere den nächsten Koeffizienten
        p = p + a[i]
}
```

Bestimmen Sie zuletzt auch die Laufzeitfunktion dieser Implementierung, sowohl genau wie auch in O-Notation!

Aufgabe 4 (P) Iteratoren

Betrachten Sie erneut die folgenden Aufgaben der vorherigen Übungsblätter. Schreiben Sie jeweils eine Funktion um das Problem zu lösen, die zum Auslesen/Bearbeiten der Elemente des Containers Iteratoren verwendet.

Hinweis: Vielleicht finden Sie den Reverse Iterator `.rbegin()` des Containers und die Methode `.insert(`*iterator*`,` *value*`)` dabei hilfreich.

a) **PlusOne.** Gegeben ist ein `vector<int> digits`, der einen positiven Integer repräsentiert. Ziffer i ist dabei an Stelle `digits[i]`, geordnet von der höchst- zur niedrigstwertigen Ziffer, die höchstwertige Ziffer steht also am Anfang der Liste. Es gibt keine führenden Nullen.

Inkrementieren Sie die repräsentierte Zahl und geben Sie den zugehörigen `vector<int>` zurück.

Bsp.: [3, 1, 2, 6] → [3, 1, 2, 7]

b) **Base-b-Addition**. Gegeben sind eine Basis b und zwei `vector<int> x, y`, die zwei positive base-b Integer repräsentieren. Ziffer i ist dabei an Stelle `i` mit Wertigkeit b^i, geordnet von der höchst- zur niedrigstwertigen Ziffer, die höchstwertige Ziffer steht also am Anfang der Liste. Es gibt keine führenden Nullen.

Addieren Sie die repräsentierten Zahlen und geben Sie die Summe als Zahl zur Basis b in einem `vector<int>` zurück.

Bsp.: [42, 53, 28], [37], 55 → [42, 54, 10]