



Aufgabe 1

3+1+2+1=7 T

Lösen von Rekurrenzen

Finden Sie für die folgenden Rekurrenzen eine geschlossene Form und beweisen Sie Ihr Ergebnis mit vollständiger Induktion.

- $g_0 = g_1 = 1, \quad g_n = 2g_{n-1} - 3g_{n-2}.$
- $g_0 = 1, \quad g_1 = -1, \quad g_n = 2g_{n-1} - g_{n-2}.$
- $g_0 = 1, \quad g_n = 3 * g_{n \text{ div } 2} + 5.$
- $g_0 = 1, \quad g_n = 3 * g_{n-1} - 1.$

Hinweise:

- Sie dürfen natürlich Ihr Ergebnis von Aufgabe 4 des letzten Blattes verwenden.
- Sollten Sie bei der Methode mit der Generierenden Funktion auf komplexwertige Wurzeln kommen, rechnen Sie einfach weiter, das Ergebnis ist dann trotzdem reell.

Aufgabe 2

2T+2T+3R+2T=6 T + 3 R

Greedy-Algorithmus

Das folgende Problem ist in der theoretischen Informatik als Rucksack-Problem bekannt:

Gegeben: k natürliche Zahlen a_1, a_2, \dots, a_k und eine natürliche Zahl b .

Gesucht: Eine Teilmenge $I \subseteq \{1, 2, \dots, k\}$ mit $\sum_{i \in I} a_i = b$.

- Nun sei zusätzlich gegeben: $a_i > \sum_{j=i+1}^k a_j$ für alle $i \in \{1, \dots, k\}$. Formulieren Sie einen Greedy-Algorithmus für das modifizierte Problem.
- Suchen Sie für jedes $k \in \mathbb{N}$ eine Instanz des allgemeinen Rucksack-Problems, die eine Lösung besitzt, welche nicht mit dem Greedy-Algorithmus aus Teilaufgabe a) gefunden werden kann.
- Implementieren Sie einen Algorithmus in Haskell, der zu einem gegebenen Rucksack eine Lösung berechnet.
- Geben Sie eine Rekurrenz für die Anzahl der Additionen Ihres Algorithmus an und finden Sie eine geschlossene Form für diese.

Aufgabe 3

2R+3R+1T+2T+3T = 6 T + 5 R

Aufwandsanalyse, Polynomarithmetik

Die folgende Definition deklariert den Datentyp Polynom:

```
newtype Polynom a = List_to_Polynom [a] deriving (Eq,Read)
```

Dabei wird ein Polynom $\sum_{i=0}^n p_i x^i$ als Liste seiner Koeffizienten gespeichert. Der Kopf der Liste soll das Element p_0 sein.

- Implementieren Sie eine Instanz für die Klasse `Show a`, die das Polynom in der Form $g_0 + g_1 x^1 + \dots$ ausgibt.
Hinweis: Hierbei ist nur die Funktion `show` zu implementieren.
- Implementieren Sie nun eine Instanz der Klasse `Num a`. Hierbei soll die Funktion `signum` das Vorzeichen des Leitkoeffizienten p_n , die Funktion `abs` den Grad des Polynoms und `fromInteger` das konstante Polynom berechnen.
- Bestimmen Sie die Anzahl der Multiplikationen und Additionen, die Ihr `(*)`-Operator benötigt um zwei Polynome vom Grad n und m zu multiplizieren.

d) Betrachten Sie den folgenden “Teile und Herrsche” Ansatz zur Multiplikation von Polynomen:

Gegeben seien die Polynome $p(x) = \sum_{i=0}^n p_i x^i$ und $q(x) = \sum_{i=0}^n q_i x^i$ vom gleichen Grad n .

Basisfall: Sind die Polynome vom Grad 0, so ist ihr Produkt $p_0 q_0$.

Teile: Sonst seien $k = n \operatorname{div} 2$, $p_0(x) = \sum_{i=0}^k p_i x^i$, $x^k p_1(x) = \sum_{i=k+1}^n p_i x^i$, $q_0(x) = \sum_{i=0}^k q_i x^i$ und $x^k q_1(x) = \sum_{i=k+1}^n q_i x^i$.

Herrsche: Multipliziere rekursiv $r_0(x) = p_0(x)q_0(x)$, $r_1(x) = p_1(x)q_0(x)$, $r_2(x) = p_0(x)q_1(x)$ und $r_3(x) = p_1(x)q_1(x)$.

Verbinde: Es ist dann $p(x)q(x) = r_0(x) + (r_1(x) + r_2(x))x^k + r_3(x)x^{2k}$

Wieviele Additionen und Multiplikationen werden benötigt? Stellen Sie jeweils eine Rekurrenz auf und geben Sie eine geschlossene Form an.

e) Betrachten Sie nun die folgende Modifikation (Karatsuba-Algorithmus):

Teile: Bleibt gleich.

Herrsche: Multipliziere rekursiv $r_0(x) = p_0(x)q_0(x)$, und $r_3(x) = p_1(x)q_1(x)$. Berechne $s_0(x) = p_0(x) + p_1(x)$ und $s_1(x) = q_0(x) + q_1(x)$, sowie rekursiv $t_0(x) = s_0(x)s_1(x)$ und anschliessend $t_1(x) = t_0(x) - r_0(x) - r_3(x)$.

Verbinde: Nun ist $p(x)q(x) = r_0(x) + t_1(x)x^k + r_3(x)x^{2k}$.

Bestimmen Sie auch hier jeweils eine Rekurrenz für die Anzahl der Additionen und Multiplikationen, sowie eine geschlossene Form.

Aufgabe 4

4+6 = 10 R

Haskell, Karatsuba, Pollard-Rho

a) Implementieren Sie den Algorithmus von Karatsuba aus Aufgabe 3 e) und vergleichen Sie die Laufzeit mit der, der klassischen Faltung.

Hinweis: Eine einfache Form der Laufzeitsmessung erhalten Sie indem Sie in der GHCi-Kommandozeile `:set +s` eingeben.

b) Implementieren Sie den Algorithmus von Pollard-Rho.