



Lösung zu Aufgabe 1

a) Raten führt hier wohl nicht zum Ziel, daher die Methode mit den "generating functions".

- *Schritt 1:* Unter Verwendung von $g_{-2} = g_{-1} = 0$ ergibt sich:

$$g_n = 2g_{n-1} - 3g_{n-2} + [n = 0] - [n = 1]$$

- *Schritt 2:* Multiplizieren mit z^n und Summieren über alle n ergibt:

$$\sum_n g_n z^n = \sum_n 2g_{n-1} z^n - \sum_n 3g_{n-2} z^n + 1 - z$$

Nun noch die Indextransformation und einsetzen:

$$G(z) = 2zG(z) - 3z^2G(z) + 1 - z$$

Auflösen nach $G(z)$:

$$G(z) = \frac{1 - z}{1 - 2z + 3z^2}$$

- *Schritt 3:* Ein Partialbruchzerlegung ergibt dann:

$$\frac{1 - z}{1 - 2z + 3z^2} = \frac{1}{2} \left(\frac{1}{1 - \frac{2}{\alpha}z} + \frac{1}{1 - \frac{2}{\hat{\alpha}}z} \right) \text{ mit } \alpha = \frac{1}{3} + \frac{\sqrt{-2}}{3} \text{ und } \hat{\alpha} = \frac{1}{3} - \frac{\sqrt{-2}}{3}$$

- *Schritt 4:* Einsetzen von

$$\frac{a}{(1 - pz)^{m+1}} = \sum_{n \geq 0} \binom{m+n}{m} a p^n z^n.$$

und ausmultiplizieren ergibt für das n -te Glied:

$$g_n = \frac{1}{2}((1 - \sqrt{-2})^n + (1 + \sqrt{-2})^n)$$

Beweis durch vollst. Induktion: $g_0 = 1 = \frac{1}{2}(1 + 1)$

$$g_1 = 1 = \frac{1}{2}(1 - \sqrt{-2} + 1 + \sqrt{-2})$$

I.H.: Die Behauptung gelte für alle $k \leq n$

$$g_{n+1} = 2g_n - 3g_{n-1} \tag{1}$$

$$(I.H.) = 2 \frac{1}{2}((1 - \sqrt{-2})^n + (1 + \sqrt{-2})^n) - 3 \frac{1}{2}((1 - \sqrt{-2})^{n-1} + (1 + \sqrt{-2})^{n-1}) \tag{2}$$

$$\tag{3}$$

- Raten ist hier einfach: $g_n = -2n + 1$ Beweis durch vollständige Induktion über n : I.A.: $g_0 = 1 = 2 * 0 + 1$
 $g_1 = -1 = -2 + 1$
 I.H.: Die Behauptung gelte für alle $k \leq n$
 $(n \rightarrow (n+1))$: $g_{n+1} = 2g_n - g_{n-1} = 2(-2n+1) - (-2(n-1)+1) = -4n+2+2n-2-1 = -2n-1 = -2(n+1)+1$
- Auch hier kann man raten: $g_n = \frac{7}{2} 3^{\lfloor \log_2 n \rfloor} - \frac{5}{2}$ Beweis durch vollständige Induktion über n : I.A.: $g_0 = 1 = \frac{7}{2} - \frac{5}{2}$
 I.H.: Die Behauptung gelte für alle $k \leq n$
 $(n \rightarrow (n+1))$: $g_{n+1} = 3g_{(n+1) \text{ div } 2} + 5$

Lösung zu Aufgabe 2

a) greedy 0 xs b liefert das Gewünschte.

```

|x+s > b = greedy s xs b
|otherwise = x:greedy (x+s) xs b

```

- b) Wähle $a_1 = 4$, $a_2 = 5$ $a_i = 1$ für $i > 2$ und $b = 5$. Für $n < 2$ funktioniert der Algorithmus immer!
- c) Probiere einfach alle Teilmengen aus!

```

potenzListe :: [a] -> [[a]]
potenzListe [] = [[]]
potenzListe (x:xs) = (potenzListe xs) ++ (map (x:) (potenzListe xs))

rucksack :: [Integer] -> Integer -> Bool
rucksack xs b = b `elem` (map sum (potenzListe xs))

```

- d) Es gibt zu einer Menge mit n Elementen jeweils $\binom{n}{k}$ Teilmengen mit k Elementen. Für jede dieser Mengen berechnet der Algorithmus die Summe (im schlimmsten Fall) und braucht dafür $k - 1$ Additionen. Schlimmstenfalls braucht der Algorithmus somit $T(n) = \sum_{k=0}^n (k - 1) \binom{n}{k}$ Additionen.

Lösung zu Aufgabe 3

a)

```

instance Show a => Show (Polynom a) where
  show (List_to_Polynom xs) = showPoly 0 xs
  where showPoly i [] = ""
        showPoly i [x] = (show x) ++ "X^" ++ (show i)
        showPoly i (x:xs) | i == 0 = (show x) ++ " + " ++ (showPoly (i+1) xs)
                          | i == 1 = (show x) ++ "X + " ++ (showPoly (i+1) xs)
                          | otherwise = (show x) ++ "X^" ++ (show i) ++ " + " ++
                                         (showPoly (i+1) xs)

```

b)

```

myzipWith f [] ys = map (f 0) ys
myzipWith f xs [] = map (\x -> f x 0) xs
myzipWith f (x:xs) (y:ys) = (f x y): myzipWith f xs ys

falte :: Num a => [a] -> [a] -> [a]
falte xs [] = []
falte xs ys = (sum (zipWith (*) (reverse xs) ys)) : falte xs (tail ys)

instance Num a => Num (Polynom a) where
  (List_to_Polynom p) + (List_to_Polynom q) = List_to_Polynom (myzipWith (+) p q)
  (List_to_Polynom p) - (List_to_Polynom q) = List_to_Polynom (myzipWith (-) p q)
  (List_to_Polynom p) * (List_to_Polynom q) = List_to_Polynom (falte p ((replicate ((length p)-1) 0) ++ q))
  fromInteger i = List_to_Polynom [fromInteger i]
  abs (List_to_Polynom p) = List_to_Polynom [fromIntegral (length p)]
  signum (List_to_Polynom p) = List_to_Polynom [signum (last p)]

```

- c) Der Algorithmus hier ist die klassische Faltung und benötigt $\Theta(mn)$ Multiplikationen Die Anzahl der Additionen, sieht man von versteckten Additionen in `length` mal ab, ist auch $\Theta(mn)$.
- d) $T_m(0) = 1$, $T_m(n) = 4 * T_m(n \text{ div } 2)$ ist die Rekurrenz für die Multiplikationen und $T_a(0) = 0$, $T_a(n) = T_a(n \text{ div } 2) + 3$ die Rekurrenz der Additionen.
 Als Lösung ergibt sich dann $T_m(n) = 4 * 2^{\lceil \log_2(2n) \rceil}$ und $T_a(n) = 3 * \lceil \log_2 \rceil$.
- e) Die Rekkurrenzen sind hier dann $T_m(0) = 1$, $T_m(n) = 3 * T_m(n \text{ div } 2)$ und $T_m(n) = 3 * 2^{\lceil \log_2(2n) \rceil} \in \Theta(x^{\log_2 3})$ und für die Additionen $T_a(0) = 0$, $T_a(n) = T_a(n \text{ div } 2) + 6$
 $T_a(n) = 6 * \lceil \log_2 \rceil$