2. Übungsaufgabe zu
Fortgeschrittene Funktionale Programmierung
Thema: Memoization - Funktionen mit Gedächtnis
ausgegeben: 27.04.2005, fällig: 04.05.2005

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP2.hs` in Ihrem Homeverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Aus der Mathematik sind folgende Beziehungen bekannt:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

$$\binom{n}{k} = \binom{n}{n-k} \tag{2}$$

  - Schreiben Sie eine Haskell-Rechenvorschrift `sumBinom :: Integer -> Integer`, so dass gilt:

$$\texttt{sumBinom}(n) = \begin{cases} \sum_{i=0}^{n} \binom{n}{i} & \text{falls } n \geq 0 \\ 0 & \text{sonst} \end{cases}$$

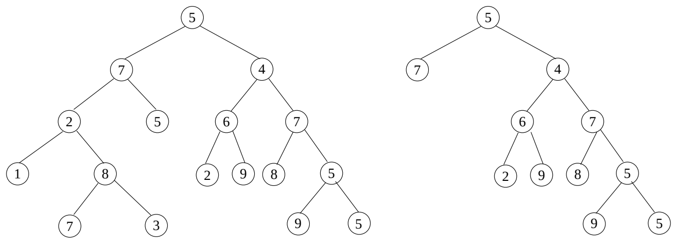
    Berechnen Sie dabei die auftretenden Binomialkoeffizienten gemäß der in (1) angegebenen Beziehung, insbesondere also unter direkter Abstützung auf die Fakultätsfunktion `fak :: Integer -> Integer`. Die Funktion zur Berechnung der Binomialkoeffizienten soll dabei mit `binkoeff` bezeichnet werden und die Signatur `binkoeff :: Integer -> Integer -> Integer` haben, wobei bei der Berechnung des Binomialkoeffizienten von "n über k" mithilfe von `binkoeff` der Wert von n als erstes, der von k als zweites Argument übergeben wird.
  - Schreiben Sie jetzt eine Haskell-Rechenvorschrift `memo_sumBinom :: Integer -> Integer`, die dasselbe Ein-/Ausgabeverhalten wie die Funktion `sumBinom` aufweist und die Berechnung auch in der gleichen Art und Weise vornimmt, wobei aber die Funktion `fak` durch eine gedächtnisbehaftete Variante `memo_fak :: Integer -> Integer` ersetzt ist.

- Schreiben Sie nun eine Haskell-Rechenvorschrift `allmemo_sumBinom :: Integer -> Integer`, in der zusätzlich zu den Werten der Fakultätsfunktion auch die Werte der bei der Summation auftretenden Binomialkoeffizienten memoriert werden. Nutzen Sie dazu die Beziehung (2) von oben aus und ersetzen Sie die Rechenvorschrift `binkoeff` durch eine Rechenvorschrift `memo_binkoeff :: Integer -> Integer -> Integer`.
- Überlegen Sie sich zum Abschluss, wie der Wert der Funktion `sumBinom` gänzlich ohne Rückgriff auf Fakultät und Binomialkoeffizienten berechnet werden könnte. Schreiben Sie dazu eine entsprechende Haskellrechenvorschrift `sb :: Integer -> Integer`.

• In dieser Aufgabe betrachten wir folgenden polymorphen Typ von Binärbäumen:

```
data Tree a = Leaf a | Node a (Tree a) (Tree a)
```

- Schreiben Sie eine Haskellrechenvorschrift `maxDepthLeaves :: Eq a => Tree a -> [a]`, die die Benennungen der tiefsten Blätter des Argumentbaums in einer Liste ausgibt von links nach rechts ausgibt, d.h., die Benennung des linkesten tiefsten Blatts ist das erste Element der Resultatliste, die des rechtesten tiefsten Blattes das letzte Element der Liste. Angewendet auf den linken Baum der nachstehenden Abbildung soll die Funktion `maxDepthLeaves` also die Resultatliste `[7,3,9,5]` liefern, angewendet auf den rechten Baum die Resultatliste `[9,5]`.



  Stützen Sie die Funktion `maxDepthLeaves` auf eine rekursive Haskellrechenvorschrift `depth :: Tree a -> Integer` zur Berechnung der Tiefe eines Binärbaums ab.
- Überlegen Sie sich, ob in Ihrer Lösung aus der vorigen Teilaufgabe Teile mehrfach berechnet werden und sich ihre Performanz durch Memorierung der entsprechenden Ergebnisse verbessern ließe. Wenn ja, schreiben Sie eine entsprechende Haskellrechenvorschrift `memo_maxDepthLeaves :: Eq a => Tree a -> [a]`.