Graphen

Definitionen

Ein *(ungerichteter) Graph* G ist ein Paar $G = (E, K)$ mit einer endlichen Menge von *Ecken* E und einer endlichen Menge von *Kanten* $K \subseteq \{\{e_i, e_j\} \,|\, e_i, e_j \in E, e_i \neq e_j\}$. Sind Schlingen $\{e_i, e_i\}$ und *Mehrfachkanten* zugelassen, so heißt G *Multigraph*. Die Kante $\{e_i, e_j\}$ wird häufig $e_i e_j$ geschrieben.
Ein *(gerichteter) Graph* G ist ein Paar $G = (E, K)$ mit einer endlichen Menge von *Ecken* E und einer endlichen Menge von *Kanten* $K \subseteq \{(e_i, e_j) \,|\, e_i, e_j \in E\} \subseteq E \times E$. Eine Kante (e_i, e_j) wird als von e_i nach e_j gerichtet betrachtet. Gilt $K \subseteq \{(e_i, e_j) | e_i, e_j \in E, e_i \neq e_j\} \subset E \times E$, so heißt der Graph *schlicht*. Auch hier kann man Multigraphen betrachten.
Ist $e_i e_j \in K$, so heißen die Ecken *benachbart* oder *adjazent*. Ist $k \in K$ und $e \in k$, so heißen k und e *inzident* und e heißt *Endecke* von k. Zwei Kanten heißen *inzident*, wenn sie eine gemeinsame Endecke besitzen. Die *Menge aller Nachbarn* $N(e)$ einer Ecke e besteht aus allen Ecken von G, die zu e *benachbart* sind. $g(e) = |N(e)|$ heißt der *Grad von* e. Ist $g(e) = 0$, so heißt e isoliert.
Bei gerichteten Graphen spricht man auch von *Anfangsecke* und *Endecke* einer Kante, von *Vorgängermenge* und *Nachfolgermenge* einer Ecke e und vom *Eingangsgrad* und *Ausgangsgrad* einer Ecke e.
Eine Folge von Kanten $k_1, \ldots, k_n$ heißt *Kantenzug* auch *Pfad*, falls es eine Folge von Ecken $e_0, \ldots, e_n$ gibt, so dass $k_i = e_i e_j \forall i \in \{1, \ldots, n\}$ ist. Sind alle k_i in einem Kantenzug verschieden, so spricht man von einem *Weg* (oder *Zyklus*). Ist $e_0 = e_n$, so heißt der Weg *geschlossen*. Ein Weg heißt *einfach*, falls alle Ecken verschieden sind. Ein (ungerichteter) Graph heißt *zusammenhängend*, falls es für jedes Paar von Ecken einen Weg zwischen ihnen gibt.

Spezifikation des ADT Graph

a) Informelle oder modellierende Methode (Beschreibung durch Text)

Sei Graph ein Netz von Buchstaben mit folgenden Operationen:

```
1  neu:: Graph
2  -- Vor.: keine
3  -- Eff.: erzeugt einen neuen leeren Graphen
4
5  istleer:: Graph -> Bool
6  -- Vor.: es existiert ein Graph
7  -- Eff.: Liefert True, wenn der Graph keine Elemente enthaelt, also
       die Knotenmenge leer ist, False sonst
8
9  kantehinzufuegen:: Kante -> Graph -> Graph
10 --Vor.:  es existiert ein Graph
```

```
-- Eff.: Fuegt die Kante in die Kantenmenge hinzu. Existiert mind.
    einer der beiden inzidierenden Knoten nicht, so passiert nichts

loeschen:: Knoten -> Graph -> Graph
-- Vor.:  Keine
-- Eff.: Entfernt den Knoten aus der Knotenmenge, sowie alle mit
    dem Knoten verbundenen Kanten. Ist der Knoten nicht enthalten,
    passiert nichts
```

b) algebraische-axiomatische Methode (Beschreibung durch Formeln und Axiome)

später

Implementierung

```
type Knoten = Char
newtype Knotenmenge = Knot [Char]
type Kante = (Knoten,Knoten)
newtype Kantenmenge = Kant [Kante]

instance Show Knotenmenge where
  show (Knot [x]) = show x
  show (Knot (x:xs)) = show x ++ "," ++ show (Knot xs)

instance Show Kantenmenge where
  show (Kant [x]) = show [x]
  show (Kant (x:xs)) = show x ++ "," ++ show (Kant xs)

newtype Graph = G (Knotenmenge,Kantenmenge)

instance Show Graph where
  show (G (a,b)) = "Knoten:␣{" ++ show a ++ "}␣Kanten:␣{" ++ show b
      ++ "}"

-- Beispielgraph g1 = {'A','B','C'},{('A','B'),('A','C')}
g1 = G (Knot ['A','B','C'],Kant [('A','B'),('A','C')])

neu:: Graph
neu = G (Knot [],Kant [])

istleer:: Graph -> Bool
istleer (G (Knot [],Kant [])) = True
istleer _ = False

-- grapherzeugen:: [Knotenmenge] -> [Kantenmenge] -> Graph
-- Übung
```

```
31
32 kantehinzufuegen:: Kante -> Graph -> Graph
33 kantehinzufuegen k (G (Knot a,Kant b)) | (elem k b) = G (Knot a,
       Kant b)
34                                        | otherwise = G (Knot a,Kant
                                              (k:b))
35
36 -- istverbunden:: Knoten -> Knoten -> Bool
37 --
38
39 -- loeschen:: Knoten -> Graph -> Graph
40
41 -- nachbar:: Knoten -> Graph -> [Knoten]
42 -- liefert eine Liste aller benachbarter Knoten zu einem gegebenen
       Knoten
43
44 -- alleKnoten::
45 -- liefert die Liste aller Knoten eines Graphen
46
47 -- grad:: Graph -> [(Knotenmenge,Int)]
48 -- grad
49
50 main = do putStrLn "Ein␣Programm␣fuer␣ungerichtete␣Gaphen:\n"
51           print (g1)
52           putStrLn"\n"
53           putStrLn "Kante␣('B','C')␣wird␣hinzugefügt!␣=>"
54           print (kantehinzufuegen ('B','C') g1)
```

Aufgabe: Implementiere die fehlenden Funktionen.