8.7 vorgegebene Funktionen

```
---------------------------------------------------------------------------
-- Auszug aus -- Standard Library: Array.hs -- -- Passend zu Hugs 1.4.
---------------------------------------------------------------------------

-- erzeugt Arrays -- array (untere Grenze, obere Grenze)
[(Index,Wert)] -- !Achtung Indizes muessen eindeutig sein array :: Ix
a => (a,a) -> [(a,b)] -> Array a b

-- erzeugt Arrays -- listArray (untere Grenze, obere Grenze) [Liste]
listArray :: Ix a => (a,a) -> [b] -> Array a b

-- Infix: Arrayzugriff (!)  :: Ix a => Array a b -> a -> b

-- Schranken des Arrays: bounds :: Ix a => Array a b -> (a,a)

-- Liste aller Indizes des Arrays: indices :: Ix a => Array a b -> [a]

-- Liste aller Elemente des Arrays: elems :: Ix a => Array a b -> [b]

-- [(Index,Wert] passend zum Array: assocs :: Ix a => Array a b ->
[(a,b)]

-- Array aenderung: array [(Index,Wert) -- !!Achtung Indizes muesen
eindeutig sein (//) :: Ix a => Array a b -> [(a,b)] -> Array a b
```

Beispiel 6. Eine 4×4-Matrix mit False-Einträgen kann folgendermaßen erzeugt werden:

```
array ((1,1),(4,4)) [((i,j),False) | i <- [1..4], j <- [1..4]]
```

Erzeugen Sie geeignete Eingaben, um Ihr Programm zu testen und sorgen Sie dafür, daß es mit den auf den WWW-Seiten des Praktikums gegebenen Tests geprüft werden kann.
.

9 Aufgabe 3: Verwalten, Erzeugen und Testen partieller Ordnungen in Java

Analog zu der Implementierung in Haskell soll auch eine Implementierung der diversen Operationen auf Relationen in Java vorgenommen werden. Zunächst sollen zwei dem Datentyp `Tord` entsprechende Klassen implementiert werden, eine mit Inzidenzlisten, die

andere mit einer Inzidenzmatrix. Die Klassen heißen `ILTord` (mit Inzidenzlisten) und `IMTord` (mit Inzidenzmatrix). Beide sind von der abstrakten Basisklasse `Tord` abgeleitet, die im folgenden dargestellt ist:

```
/* -*-java-*-
 *
 * Datei: Tord.java
 */

/*
 * Abstrakte Basisklasse für Tord-Implementierungen
 * mit Inzidenzliste und Inzidenzmatrix.
 */
abstract class Tord {
  /*
   * Liefert eine Liste (Array) aller in einer gegebenen Relation
   * angeordneten Elemente.
   */
  public abstract int[]   elements();

  /*
   * Prüft ob ‘a’ < ‘b’ in der gegebenen Relation gilt.
   */
  public abstract boolean lessThan(int a, int b);
};
```

Im Rahmen der vorliegenden Aufgabe werden zunächst nur Relationen über `int`–Werten betrachtet. Als Eingabe für die verschiedenen Programme soll eine Textdatei verwendet werden, in der eine binäre Relation zwischen `int`–Werten wie folgt definiert ist. Jede Zeile der Eingabedatei ist entweder leer oder enthält genau zwei, durch Leerzeichen getrennte Zahlen. So würde die folgende Datei:

```
1 2
100  3

45  789
```

die Relation $\{(1,2),(100,3),(45,789)\}$ definieren. Um eine solche Datei einzulesen steht die Klasse `IntRel2` zur Verfügung:

```
class IntRel2 {
    static public int[][]  readFile(String name);
}
```

Mit Hilfe der Anweisung `int[][] rel = IntRel2.readFile(''abc'');` würde die Datei `abc` eingelesen und die in ihr definierte Relation in Form eines Arrays aus zweielementigen `int`–Arrays zurückgeliefert. Wenn wir bei obigem Beispiel bleiben wäre das Ergebnis analog zu: `int[][] rel = {{1, 2}, {100, 3}, {45, 789}};`.

Repräsentation durch Inzidenzlisten

In dieser Aufgabe soll die Klasse `ILTord` vollständig implementiert werden:

```
/* -*-java-*-
 *
 * Datei: ILTord.java
 */

// Instanzen dieser Klasse repräsentieren Inzidenzlisten.
class IList {
  public int    element;
  public Vector greaterOrEqual;

  public IList() {
    element = 0;
    greaterOrEqual = new Vector(5);
  }
}

// Instanzen repräsentieren Relationen mittels Inzidenzlisten.
class ILTord extends Tord {
  // Die Inzidenzlisten für die Elemente der Relation.
  IList[] relation;

  public ILTord(int[][] ipairs) {
    // Implementieren: Aufbau der Inzidenzlisten,
    // Berechnung der transitiven Hülle.
  }

  public int[] elements() {
    // Implementieren: Ein 'int' Array mit den Elementen
    // der Relation erzeugen.
  }

  public boolean  lessThan(int a, int b) {
    // Implementieren: Prüfen ob 'a' < 'b' gilt.
  }

  // Bei Bedarf weitere Elemente und Methoden.
```



```
}
```

Benutzen Sie die Klasse `IList` zur Darstellung der Inzidenzlisten zu den jeweiligen Elementen der Relation. Überlegen Sie, wie sich Mengen in Java darstellen lassen, schauen Sie sich dazu die Standardklasse `Hashtable` an.

Repräsentation durch Inzidenzmatrizen

Implementieren Sie eine Darstellung von Relationen durch Inzidenzmatrizen, indem sie die folgende Klasse vervollständigen:

```
/* -*-java-*-
 *
 * Datei: IMTord.java
 */

// Instanzen dieser Klasserepräsentieren Relationen
// mittels Inzidenzmatrizen.
class IMTord extends Tord {
  // Datenstruktur zur Speicherung der Inzidenzmatrix.
  boolean[][] matrix;

  public IMTord(String file) {
    // Implementieren: Inzidenzmatrix aufbauen,
    // transitive Hülle berechnen.
  }

  public int[] elements() {
    // Implementieren: Ein 'int' Array mit den Elementen
    // der Relation erzeugen.
  }

  public boolean lessThan(int a, int b) {
    // Implementieren: Prüfen ob 'a' < 'b' gilt.
  }

  // Bei Bedarf weitere Elemente und Methoden.
}
```

Ordnungsprüfungen

Implementieren sie die Funktionen `ist_zyklisch`, `ist_verband` und `ist_linear` in der unten abgebildeten Klasse `Aufgabe`.

```
/* -*-java-*-
 *
 * Datei: Aufgabe.java
 */

class Aufgabe {

  static public boolean ist_zyklisch(Tord t) {
    // Implementieren.
  }

  static public boolean ist_verband(Tord t) {
    // Implementieren.
  }

  static public boolean ist_linear(Tord t) {
    // Implementieren.
  }

  public static void main(String[] args) {
    Tord    t;

    t = new ILTord(IntRel2.readFile(args[0]));
    System.out.println(ist_zyklisch(t));
    System.out.println(ist_verband(t));
    System.out.println(ist_linear(t));

    t = new IMTord(IntRel2.readFile(args[0]));
    System.out.println(ist_zyklisch(t));
    System.out.println(ist_verband(t));
    System.out.println(ist_linear(t));

    return;
  }
}
```

Durch Angabe der Eingabedatei für die Relation beim Aufruf (z.B.: `java Aufgabe abc`) können Sie so ihre Lösung für verschiedene Relationen verifizieren.

Topologisches Sortieren

Implementieren sie ein in der Größe der Eingabe lineares Verfahren zur topologischen Sortierung. Vervollständigen Sie dazu folgende Klasse:

```
/* -*-java-*-
```



```
 *
 * Datei: Topsort.java
 */

class Topsort
{

   int[] tsort(int[][] ipairs) {
     // Implentieren.
   }

   static public void main(String[] args) {
     System.out.println(tsort(IntRel2.readFile(args[0])));
   }
}
```

Testen Sie auch hier Ihr Programm mit geeigneten Eingaben.

Literatur

[Abr90] S Abramsky. Finding fixpoints in abstract interpretation. Technical Report DOC 90/20, Dept of Computing, Imperial College, 1990.

[BW88] Richard Bird and Philip Wadler. *Introduction to Functional Programming.* Prentice-Hall International, London, 1988.

[Cun94] H. Conrad Cunningham. Notes on functional programming with gofer. Technical Report UMCIS-1994-01, Dept of Computer and Information Science, University of Mississippi, January 1994.

[Dav92] A.J.T. Davie. *An Introduction to Functional Programming Systems Using Haskell.* Cambridge University Press, Cambridge, 1992.

[Hen80] Peter Henderson. *Functional Programming: Application and Implementation.* Prentice-Hall International, London, 1980.

[Her92] Helmut Herold. *UNIX und seine Werzeuge, Band 4: Lex und Yacc.* Addison-Wesley, Bonn, 1992.

[Hin92] Ralf Hinze. *Einführung in die funktionale Programmierung mit Miranda.* Teubner, Stuttgart, 1992.

[Hol91] Ian Holyer. *Functional Programming with Miranda.* Pitman, 1991.

[Klo97] N. Klose. Implementierung der n-pack transformation im rahmen eines frontends für haskell. Master's thesis, J.W.Goethe Universität, Frankfurt, 1997. in preparation.