```
                (return (a:w)) >>= wortaux

main = do
        x <- wort
        print x
```

- **referentielle Transparenz**
  Bisher haben wir noch nicht betrachtet, in welcher Weise der Datentyp `IO a` definiert ist, und wie die beiden Monadenfunktion auf ihm operieren.
  Der Typ `IO a` bleibt in Haskell abstrakt, d.h. seine Definition ist dem Programmierer nur über die auf ihm definierten Funktionen zugänglich, nicht aber um mittels Pattern-Matching Objekte dieses Typs zu dekomponieren.

8 Aufgabe 2: Verwalten, Erzeugen und Testen partieller Ordnungen in Haskell

8.1 Einige mathematische Begriffe

Diese und noch weitere Begriffe finden sich auch in [SS].

Relation

Eine zweistellige (binäre) Relation R auf einer Menge S ist eine Teilmenge von $S \times S$, d.h. geordnete Paare (s_1, s_2), s_1 und s_2 aus S, sowie $(s_1, s_2) \neq (s_2, s_1)$. Eine mögliche Darstellung ist: $s_1 R s_2$

Beispiel 3. Sei $S = 2, 3, 5, 6$ und sei die Relation R definiert durch "ist Teiler von". Dann stellt R die folgende Menge dar: $\{(2,2), (2,6), (3,3), (3,6), (5,5), (6,6)\}$

partielle Ordnung

Eine Relation R auf einer Menge S heißt (partielle) Ordnung, wenn für alle a, b, c aus S gilt:

1. R ist *reflexiv*:
$$aRa$$
2. R ist *antisymmetrisch*:
$$aRb \wedge bRa \iff a = b$$

3. R ist *transitiv*:
$$aRb \wedge bRc \Rightarrow aRc$$

Eine partielle Ordnung nennt man auch reflexive Ordnung.

Beispiel 4. $\leq$ als Relation auf der Menge der natürlichen Zahlen

Antisymmetrie, Zyklen

Die Antisymmetrie zusammen mit der Transitivität bedingt, daß es in einer Anordung $a_1 R a_2 R a_3 R \ldots$ von Elementen aus S gemäß der Ordnung keine Zyklen geben darf.

Beispiel 5. Ein Beispiel für einen Zyklus ist $a \leq b \leq c \leq a$.

Stellt man die Ordnung als (gerichteten) Graph dar, wobei die Knoten Elemente aus S repräsentieren, und eine Kante von Knoten 1 nach Knoten 2 bedeutet, daß die beiden zugehörigen Elemente ein (geordnetes) Paar in der Relation R aus $S \times S$ bilden, so kann es in dem Graph keinen in sich geschlossenen Weg (loop, Schleife) geben.

transitive Hülle

Nimmt man nun neue Kanten derart hinzu, daß man von einem Knoten aus Kanten zu all den Knoten erzeugt, die auf einem Pfad im Graph erreichbar sind (entspricht der Transivität), so erhält man die transitive Hülle $H(R)$ der Relation R.

Eine Relation R heißt vollständig bzw. linear, wenn für alle (a, b) aus $S \times S$ gilt: $(a, b) \in R$ oder $(b, a) \in R$ oder $a = b$.

Das bedeutet, daß zwei beliebige Elemente aus S bzgl. der gegebenen Relation zueinander in Beziehung stehen. Übertragen auf den Ordnungsbegriff bedeutet es, daß sich zwei beliebige Elemente aus S anordnen lassen, d.h. eines der beiden tritt bzgl. der Ordnung vor dem andern auf bzw. "ist größer".

Verband

Sei nun M eine Menge und R eine reflexive Ordnung auf M. M ist ein Verband, wenn gilt: Für jede Teilmenge von M gibt es genau eine eindeutige kleinste obere Schranke und genau eine größte untere Schranke.

topologische Sortierung

Die topologische Sortierung topsort(G) eines gerichteten azyklischen Graphen $G = (V, E)$ sortiert die Knoten (gemäß einer linearen Ordnung) wie folgt: Falls es eine Kante von Knoten i nach Knoten j gibt, so wird Knoten i vor Knoten j einsortiert. Durchläuft man die Knoten in der durch topsort(G) erzeugten Reihenfolge, so wird ein Knoten erst dann betrachtet, wenn alle seine Vorgänger betrachtet wurden. Dies ist z.B. dann von Bedeutung, wenn die Kanten einen Informationsfluß von Knoten zu Knoten repräsentieren. Ein Knoten kann dann erst ausgewertet werden, wenn alle seine Eingabeknoten ausgewertet sind.

Pseudocode-Darstellung von topsort():

```
topsort(vertex v)
{
  falls v = Wurzel von G:
    num=1;

  markiere v;
  Ordne v die Ordnungsnummer num zu.

  für alle nachfolger w von v:
  falls w noch nicht markiert:
    topsort(w)
}
```

Warshall-Algorithmus

Der Warshall-Algorithmus berechnet die transitive Hülle. (under construction)

8.2 Aufgabenbeschreibung

In der zweiten Aufgabe sollen partielle Ordnungen verarbeitet werden. Partielle Ordnungen können durch erzeugende Relationen gegeben sein und davon wollen wir hier ausgehen. Eine Eingabedatei kann z.B. folgenden Inhalt haben:

```
[("A","B"), ("B","C")]
```

Um mit so gegebenen Ordnungen algorithmisch umgehen zu können muß man die partielle Ordnung zur erzeugenden Relation berechnen.

Der wesentliche Schritt ist die Berechnung der transitiven Hülle, mit deren Hilfe man dann die Ordnungsrelation selbst als einfache Funktion von zwei Argumenten berechnen kann. Damit kann man weitere Eigenschaften der Ordnung überprüfen.

Zur Berechnung der transitiven Hülle sollen Sie u.a. den Warshall-Algorithmus implementieren, der sukzessive die transitive Hülle berechnet.

Außerdem sollen Funktionen geschrieben und auf korrekte Funktionalität getestet werden, die folgendes überprüfen:

- Ist die erzeugte Ordnung antisymmetrisch (enthält sie Zyklen)?
- Ist die Ordnung ein Verband?
- Ist die Ordnung linear?

Ein schneller Algorithmus, der die Antisymmetrie einer Ordnung erkennt (bzw. der Zyklen in einem gerichteten Graphen entdeckt), ist das topologische Sortieren [Knu73a, Knu73b], das eine geordnete Menge in eine totale Ordnung einbettet, wenn das möglich ist. Verwenden Sie diesen für die Antisymmetrie.

8.3 Vorgehensweise

Beachten Sie, daß die Namen und Typen von Funktionen, die in dieser Aufgabenbeschreibung enthalten sind, genau vorgegeben sind, damit die Überprüfung der Funktionen durch den Tutor vereinfacht wird.

Gehen Sie bitte folgendermaßen vor.

Definieren Sie folgende Typen, so daß die Datei folgendermaßen beginnt

```
> import Array
> import List

> data Hash a
>   = Hash (a->Int) (Array Int [(a,Int)])
>     deriving (Show)

> data Ordnung
>   = Ordnung [(String,String)] (Array Int String) (Hash String)
>             (Array (Int,Int) Bool)
>     deriving Show

> data Tord a
>   = Tord [a] (a->a->Bool)
>     deriving Show

> data Set a
>   = Set [a]
>     deriving Show

> instance Eq a => Eq (Set a) where
>   xs == ys = subset xs ys && subset ys xs
```

```
> subset (Set xs) (Set ys)
>   = all (\x -> x `elem` ys) xs
```

Die Inhalte der Datenstrukturen sind:

Hash Eine Hashtabelle mit einer Hashfunktion, einem Array, das als Einträge den Wert und einen eindeutigen Index zu diesem Wert hat. Diese Hashtabelle wird vermutlich nur für Strings benötigt. Die Hashtabelle wird im Programm dazu verwendet die Strings aus der Eingabe schnell mit ihrer Nummer zu assoziieren.

Ordnung Ein Datentyp mit folgenden Attributen: Liste der erzeugenden Paare, wobei die Objekte Strings sind; Ein Array, das die eindeutige Zuordnung Index zu String macht; eine Hashtabelle, die einem String den Index zuordnet; und eine Inzidenzmatrix,

Tord Ein Datentyp, der als erstes Argument die Liste der Elemente hat auf denen die Ordnung definiert ist, und als zweites die Funktion, die unter Verwendung des Datentyps `Ordnung` zu je zwei Elementen bestimmt, ob zwischen die Ordnung gilt.

Folgende Funktionen sollen geschrieben werden:

ist_zyklisch Vom Typ `Tord a -> Bool`. Diese Funktion soll testen, ob die transitive Ordnung im ersten Argument antisymmetrisch ist, bzw. ob diese Zyklen enthält.

ist_verband Vom Typ `Tord a -> Bool`. Diese Funktion soll testen, ob die transitive Ordnung im ersten Argument ein Verband ist. Dazu genügt es, zu überprüfen, ob je zwei Elemente genau eine kleinste obere Schranke haben, ob die Bildung dieser Schranke eine assoziative Operation ist und ob die Ordnung tatsächlich eine partielle Ordnung ist.

ist_linear Vom Typ `Tord a -> Bool`. Diese Funktion soll testen, ob die transitive Ordnung im ersten Argument total ist (linear).

Die Erzeugung der Ordnung soll zunächst nur mittels Listen erfolgen und dann, als Verbesserung bzw. Optimierung, mittels Hashtabellen, Arrays und Inzidenzmatrizen.

Der Test auf Linearität soll mit zwei Methoden erfolgen, einmal mittels der Ordnung (`Tord` ...), d.h. durch den globalen Test, ob für alle Elemente x, y stets $x == y$, $x < y$ oder $y < x$ gilt. Außerdem soll dieser Test mit dem effizienten topologischen Sort ebenfalls durchgeführt werden.

8.4 Erzeugung der Ordnung mit Listen

Die Funktion soll folgenden Namen und Typ haben:

```
mk_pordl :: [(a,a)] -> Tord a
```


Als Vorgehensweise kann gewählt werden: Erzeugen einer Assoziationsliste der transitiven Hülle. Diese kann dann zum Erzeugen der <-Funktion benutzt werden. Achten Sie darauf, daß die Reflexivität nicht implizit angenommen wird.

8.5 Erzeugung der Ordnung mit Hashtabelle und Arrays

Die Funktion soll folgenden Namen und Typ haben:

```
mk_pord :: [(String,String)] -> Tord String
```

Als Eingabe beschränken wir uns auf erzeugende Paare von Strings. Es sollen zunächst die vorkommenden Strings ermittelt werden und diese eindeutig nummeriert werden. Damit man aus dem String schnell den Index ermitteln kann, soll eine Hashtabelle angelegt werden. Die Inzidenzmatrix hat als Indexbereich $[1..n] \times [1..n]$, wobei n die Anzahl der vorkommenden Strings ist.

Auf der Inzidenzmatrix ist dann das Iterationsverfahren durchzuführen. Hierbei wird eine Inzidenzmatrix $I(i, j), 1 \leq i, j \leq n$ angelegt, die die Wahrheitswerte der Aussagen $a_i < a_j$ enthält. Der Warshall-Algorithmus iteriert die Transitivitätsregel und erzeugt sukzessive neue Inzidenzmatrizen, bis sich nichts mehr ändert.

Die Regel ist: $I(i, j) := I(i, j) \vee I(i, k) \wedge I(k, j)$.

8.6 Topologisches Sortieren

```
topsort :: [(String,String)] -> (Bool,[String])
```

Gegeben ist eine Liste von erzeugenden Relationen. Zunächst macht man eine eindeutige Index-Namens-Zuordnung Man erzeugt daraufhin zwei Arrays:

- zu jedem Index eine Liste der Indizes der grösseren Elemente.
- zu jedem Index eine Liste der Indizes der kleineren Elemente.

Der Algorithmus gibt nach und nach eine Liste aus, die von oben beginnt, so daß die Ausgabe $a_1, \ldots a_n$ ist, so daß niemals $a_1 < a_2$ ist. Wenn das nicht möglich ist, dann wird der Boolesche Wert False an erster Stelle ausgegeben. Ansonsten ist der erste Wert True, und der zweite Wert die Liste $a_1, \ldots a_n$.

Der Algorithmus sollte linearen Ressourcenbedarf abhängig von der Größe der Eingabe haben. Dies kann man erreichen, wenn man das Array, das die Indizes der grösseren Elemente enthält, jeweils nach einer Ausgabe eines Elementes um dieses Element bereinigt und dann diejenigen sich merkt, die maximal sind, d.h. die keine grösseren Elemente mehr haben. Das sind diejenigen, deren Einträge nach der Änderung zur leeren Liste werden.

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