Zadanie zaliczeniowe z Haskella (8p)

Streszczenie

Zaimplementuj tablice, tj. struktury danych pozwalające na odczytanie elementu o podanym indeksie.

Tablica może być indeksowana dowolnym typem spełniającym pewne warunki.

Przy użyciu tych tablic zaimplementuj grafy i algorytmy na nich.

Dopuszczamy rozwiązania częściowe (za odpowiednio mniej punktów). Na przykład:

- za rozwiązanie części 2 i 3 za pomocą list albo tablic z Data.Array można dostać do 4p.
- za samą implementację indeksów i tablic (ale w pełni poprawną!) można dostać do 4p.

Specyfikacja

1. Napisz moduł `MyArray`, eksportujący:

- klasę indeksów (w razie potrzeby można dodać więcej metod):

```
class Ord a => Ix a where
  --| range (lo,hi) daje listę wszystkich indeksów
  --  pomiędzy lo a hi
  range :: (a, a) -> [a]
  --| index (lo,hi) i daje numer kolejny indeksu i w zakresie
  -- (od 0)
  -- np index (7,9) 8 = 1; index ('a','d') 'd' = 3
  -- komunikat o błędzie jesli indeks poza zakresem.
  index :: (a, a) -> a -> Int
  inRange :: (a, a) -> a -> Bool
  rangeSize :: (a, a) -> Int
```

- instancje tej klasy, co najmniej:

```
instance Ix Char
instance Ix Int
instance Ix Integer
instance (Ix a, Ix b) => Ix (a,b)
```

(1p za klasę indeksów i instancje)

- typ tablic `Array`, z operacjami

```
-- | Buduje tablicę dla danego zakresu i listy elementów
listArray :: (Ix i) => (i, i) -> [e] -> Array i e
-- | Daje element tablicy o podanym indeksie
(!)           :: Ix i => Array i e -> i -> e
-- | Daje listę elementów tablicy (w kolejności indeksów)
elems     :: Ix i => Array i e -> [e]
-- | Buduje tablicę z podanej listy par (indeks,wartość)
array      :: (Ix i) => (i, i) -> [(i,e)] -> Array i e
-- | Daje tablicę będącą wariantem danej, zmienioną pod podanym indeksem
update :: Ix i => i -> e -> Array i e -> Array i e
-- | Daje tablicę będącą wariantem danej, zmienioną pod podanymi indeksami
(//)         :: (Ix i ) => Array i e -> [(i,e)] -> Array i e
```

Dla dowolnej listy `es`, musi być spełniony warunek

```
elems (listArray (1,length es) es) == es
```

Pomocne testy sprawdzający powyższy warunek i jemu podobne są zawarte w pliku `ArrayTests.hs`, można je uruchomić przez `runhaskell` ze swoim modułem `MyArray`:

```
$ runhaskell ArrayTests.hs
array
+++ OK, passed 100 tests.
update
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
rangelo2
+++ OK, passed 100 tests.
rangelo3
+++ OK, passed 100 tests.
acc
+++ OK, passed 100 tests.
elems
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
```

Moduł ten:

- nie powinien korzystac z innych modułów (poza Prelude; dopuszczone jest umieszczenie klasy indeksów w osobnym module),

- powinien eksportować tylko typ Array, ale już nie jego konstruktory ,
- wszystkie operacje eksportowane przezeń muszą być polimorficzne względem typów indeksów i wartości (tylko ograniczenie (Ix i) dla indeksów).

(3p za tablice)

2. Używając tego modułu napisz moduł reprezentujący grafy (o wierzchołkach typu Int) i operacje na nich (przynajmniej te potrzebne do programu poniżej) (2p)
3. Napisz program, który wczyta graf skierowany w formacie

```
v1 v11 v12 ... v1n
...
vm vm1 vm2 ... vmk
```

(każdy wiersz zawiera numer wierzchołka i listę wierzchółków z nim sąsiadujących)

i wypisze listę wierzchołków do których istnieje ścieżka z wierzchołka 1. (2p)

Gdy program wywołany jest z argumentami, graf wczytywany jest z pliku o ścieżce podanej w pierwszym argumencie, wpp z stdin.

Należy obsłużyć możliwe sytuacje błędne (pusty plik, brak pliku). Komunikaty postaci

```
Zadanie.hs:(18,1)-(20,60): Non-exhaustive patterns in function
```

będą podstawą do obniżenia oceny.