Programowanie funkcyjne — kolokwium nr 1, 6.12.2017

Instrukcja: Rozwiązania zadań należy przesłać do godziny 11:00 na adres `kolokwium.pf@gmail.com` (decyduje data stempla googlowego). Każde zadanie należy przesłać w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia `import`. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie. Uwaga: korzystanie z internetu poza wyznaczonym czasem skutkuje automatycznym otrzymaniem 0 punktów.

Zadanie 1. Napisać funkcję rd :: Integer → [Integer], która dla podanego $n \geqslant 1$ zwraca (być może nieskończoną) listę składającą się ze wszystkich liczb k takich, że suma dodatnich dzielników liczby k równa jest $n + k$. Zatem np. rd 1 zwraca listę wszystkich liczb pierwszych, rd 2 daje listę pustą, a rd 3 to lista zawierająca tylko liczbę 4.

W przypadku, gdy wynik jest listą skończoną, wskazane jest, by funkcja kończyła działanie. Wskazówka: jeśli liczba k ma dzielnik d, to ma również dzielnik k/d. Na tej podstawie można oszacować, że dla $n > 1$ w wyniku funkcji rd n nie może być liczb większych niż n^2.

Zadanie 2. Napisać funkcję repl :: Eq $a \Rightarrow [a] \to [(a, a)] \to [a]$, która dla danej listy ℓ oraz listy par dokonuje zamian elementów w ℓ w taki sposób, że każde wystąpienie pierwszego elementu pewnej pary zostaje zamienione na drugi element tej pary, np.

```
repl [1,2,3,1,2] [(2,4)] = [1,4,3,1,4]
repl "alamakota" [('a','u'), ('o','e')] = "ulumuketu"
```

W rozwiązaniu należy w istotny sposób użyć funkcji foldl lub foldr. Uwaga: można założyć, że wszystkie elementy występujące w parach są różne, tzn. lista k par $(a_1, b_1), ..., (a_k, b_k)$ zawiera $2k$ różnych elementów $a_1, ..., a_k, b_1, ..., b_k$.

Zadanie 3. Słabym drzewem binarnym nazywamy strukturę, w której każdy wierzchołek zawiera pewną wartość, a ponadto ma 0, 1 lub 2 wierzchołki potomne. W tym ostatnim przypadku kolejność potomków nie jest rozróżniana, tzn. drzewa różniące się jedynie kolejnością potomków uznajemy za równe. Słabe drzewo binarne nie może być puste. Stworzyć typ Sdb a, przechowujący elementy typu a w słabym drzewie binarnym, i zdefiniować funkcje:

```
el :: Eq a => Sdb a -> a -> Bool
eq :: Eq a => Sdb a -> Sdb a -> Bool
sdb2list :: Sdb a -> [a]
```

Funkcje mają, odpowiednio: sprawdzać czy podany element należy do drzewa, sprawdzać czy podane drzewa są równe oraz zamieniać drzewo na listę przeszukując je wszerz (kolejność przeglądania potomków może być dowolna).