Контролно по Функционално програмиране

спец. Компютърни Науки, 20.01.2011 г.

Вариант А

Задача 1 **(1,5 т.)** Да се напише функция на Scheme (chinese? g), която за даден граф g проверява дали за него е изпълнен китайският закон: “даден възел може да има повече от един наследник само ако е той е единствен наследник на родителя си”.

Задача 2. **(1 т.)** Да се напише функция на Scheme (pairs s n), която по даден поток s: a_1, a_2, a_3, …, намира потока от двойки от елементи на потока s в следния вид: (a_1 . a_{n+1}), (a_2 . a_{n+2}), (a_3 . a_{n+3}), (a_4 . a_{n+4}), … .

Задача 3. **(2,5 т.)** Да се напишат функции на Haskell

а) count :: (Eq a) ⇒ a → [a] → Int, която намира броя на срещанията на елемент в списък

б) remove :: (Eq a) ⇒ [a] → [a], която премахва всички срещания на елемент от списък

в) hist :: (Eq a) ⇒ [a] →[(a,Int)], която по даден списък създава списък от двойки с честотата на срещанията на всеки негов елемент.

Пример: hist [1, 2, 3, 1, 5, 3] ~> [(1, 2), (2, 1), (3, 2), (5 1)]

Контролно по Функционално програмиране

спец. Компютърни Науки, 20.01.2011 г.

Вариант Б

Задача 1. **(1,5 т.)** Да се напише функция на Scheme (paths-through n g), която намира всички ациклични пътища в граф g, които минават през възела n.

Задача 2. **(1 т.)** Да се напише функция на Scheme (repeat-stream s), която по даден поток s от точкови двойки от вида (a_i . b_i), където a_i и b_i са неотрицателни цели числа, генерира нов поток, в който всяко a_i се повтаря b_i на брой пъти. Пример: Нека s е потокът (2 . 2), (0 . 1), (3 . 5), …. Тогава (repeat-stream s) трябва да генерира потока 2, 2, 0, 3, 3, 3, 3, 3, … .

Задача 3. **(2,5 т.)** Да се напишат функции на Haskell

а) startsWith :: (Eq a) ⇒ [a] → [a] → Bool, която проверява дали първият списък е начало на втория

б) substAll :: (Eq a) ⇒ [a] → [a] →[a] → [a], която заменя всяко срещане на втория списък като подсписък в първия с третия.

в) substDictionary :: (Eq a) ⇒ [a] →[([a],[a])] →[a], която за всяка двойка (x,y) от втория списък (“речник“) заменя всяко срещане на x като подсписък в първия списък с y.

Пример: substDictionary “I hate Scheme and Scheme hates me” [(“hate”,“love”), (“Scheme”,”Haskell”)] ~> “I love Haskell and Haskell loves me”