Домашна работа №3 по Функционално програмиране

Специалност “Информационни системи”, 2019/2020 учебна година

Крайният срок за предаване на домашните работи е ***03.06.2020 г.***, 23:55 ч.

Решенията трябва да са готови за компилиране и автоматично тестване. Важно е писмените работи да бъдат добре форматирани и да съдържат коментари на ключовите места.

Предайте решенията на всички задачи в един файл с наименование hw3_<FN>.hs, където <FN> е Вашият факултетен номер.

Приятна работа и успех!

Нека е дадено следното представяне на двоично дърво:

```
data BTree a = Empty | Node a (BTree a) (BTree a)
```

Задача 1. Да се дефинира функция `containsWord :: BTree Char -> String -> Bool`, която по дадено двоично дърво от символи и дума, съставена от поне една буква, проверява дали думата се среща в дървото, като последният символ от думата е лист в това дърво.

Пример:

```
t1 :: BTree Char                                        --      a
t1 = Node 'a' (Node 'c' (Node 'f' Empty Empty)          --     / \
                        (Node 'd' Empty Empty))         --    c   b
              (Node 'b' Empty                           --   / \   \
                        (Node 'e' Empty Empty))         -- f   d   e

containsWord t1 "acd" → True

containsWord t1 "cd"  → True

containsWord t1 "ac"  → False
```

Задача 2. Да се дефинира функция `genWords :: BTree Char -> [String]`, която по дадено двоично дърво от символи връща списък от всички думи, съдържащи се в него.

Пример:

```
genWords t1 → ["acf","acd","abe","cf","cd","f","d","be","e"]
```

Задача 3. Да се дефинира функция `allContain :: [BTree Char] -> [String]`, която по даден списък от двоични дървета от символи връща списък от тези думи, които се съдържат във всички дървета.

Пример:

```
t2 :: BTree Char                                    --     a
t2 = Node 'a' (Node 'c' (Node 'd' Empty Empty)      --    / \
                        Empty)                      --   c   b
              (Node 'b' Empty Empty)                --  /
                                                    -- d
allContain [t1, t2] → ["acd","cd","d"]
```

Задача 4. Нека е дадено следното представяне на n-арно дърво:
`data NTree = Nil | NNode Int [NTree]`
Да се дефинира функция `isGraceful :: NTree -> Bool`, която приема n-арно дърво и проверява дали то е *грациозно*. Казваме, че едно дърво е *грациозно*, ако абсолютните стойности на разликите между стойностите на всеки елемент и бащиния му са четни.

Пример:

```
t3 :: NTree                    --     1
t3 = NNode 1 [NNode 3 [],      --  / / \ \
              NNode 5 [],      -- 3 5   7 9
              NNode 7 [],
              NNode 9 []]
                                       --   7
t4 :: NTree                            --   |
t4 = NNode 7 [NNode 9 [NNode 5 [],     --   9
                       NNode 2 []]] --  / \
                                       -- 5   2

isGraceful t3 -> True  (|3-1|=2, |5-1|=4, |7-1|=6, |9-1|=8)
isGraceful t4 -> False (|9-7|=2, |5-9|=4, |2-9|=7)
```

БОНУС Задача 5 (първите 4 задачи са достатъчни за оценка 6,00). Да се дефинира функция `isBinarySearchTree :: BTree Int -> Bool`, която проверява дали дадено двоично дърво от цели числа е *двоично дърво за търсене*. Казваме, че едно двоично дърво е *двоично дърво за търсене*, ако лявото му поддърво съдържа само възли със стойности, по-малки от тази в корена, а дясното

му поддърво - само стойности, по-големи или равни на тази в корена. Освен това, трябва и самите поддървета също да са *двоични дървета за търсене*.

Примери:

```
t5 :: BTree Int                                  --       8
t5 = Node 8 (Node 3 (Node 1 Empty Empty)         --     /   \
                    (Node 4 Empty Empty))        --    3     10
            (Node 10 (Node 9 Empty Empty)        --   / \    / \
                     (Node 14 Empty Empty)) -- 1   4  9   14

isBinarySearchTree t5 → True

t6 :: BTree Int                                  --       8
t6 = Node 8 (Node 3 (Node 1 Empty Empty)         --     /   \
                    (Node 4 Empty Empty))        --    3     10
            (Node 10 (Node 5 Empty Empty)        --   / \    / \
                     (Node 14 Empty Empty)) -- 1   4  5   14

isBinarySearchTree t6 → False (в дясното поддърво има стойности,
                                по-малки от корена)

t7 :: BTree    Int                               --       8
t7 = Node 8 (Node 3 (Node 5 Empty Empty)         --     /   \
                    (Node 6 Empty Empty))        --    3     10
            (Node 10 (Node 5 Empty Empty)        --   / \    / \
                     (Node 14 Empty Empty)) -- 5   6  9   14

isBinarySearchTree t7 → False (лявото поддърво не е BST)
```