Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Домашна работа 3

курс Функционално програмиране
за специалности Информатика и Компютърни науки (2-ри поток)
зимен семестър 2020/21 г.

Редакции

31.12.2020 г. – В дефиницията на функцията `values` имаше грешка. Типът на функцията беше записан като `values :: a => Strategy -> (Tree a) -> [a]`, а коректното е `values :: Strategy -> (Tree a) -> [a]`.

01.01.2021 г. – Добавено е уточнение към точки Г) и Д), относно PPM формата.

08.01.2021 г. – Коригирано е изискването за имената на файловете, които трябва да се предадат – `Task1.hs` и `Task2.hs`, вместо `1.hs` и `2.hs`. Добавена е и информация за модулите, в които трябва да се сложат решенията и тестовете

Изисквания

Освен на изискванията, които са описани в документа “Схема за оценяване”, решенията ви трябва да отговарят и на следните:

- Задачите трябва да се решат на Haskell.
- Към функциите, които сте написали трябва да изготвите подходящи unit test-ове.
- Решението на всяка задача трябва да бъде в отделен модул, в отделен файл. Имената на модулите за първа и втора задача трябва да бъдат съответно `Task1` и `Task2`. Имената на файловете трябва да бъдат `Task1.hs` и `Task2.hs`
- Unit test-овете към задачите сложете в модули `Task1_Test` и `Task2_Test`. Имената на файловете трябва да бъдат `Task1_Test.hs` и `Task2_Test.hs`
- Разширението на файловете трябва да отговарят на приетите правила за Haskell, според стила, който се използвали. Ако използвате “Literate programming”, то трябва да бъде `.lhs`, а в противен случай – `.hs`.

Задача 1

Двоично дърво, съхраняващо елементи от тип `a`, ще представяме чрез следния тип в Haskell:

```
data Tree a = EmptyTree | Node {
                            value :: a,
                            left  :: Tree a,
                            right :: Tree a
                          } deriving (Show,Read)
```

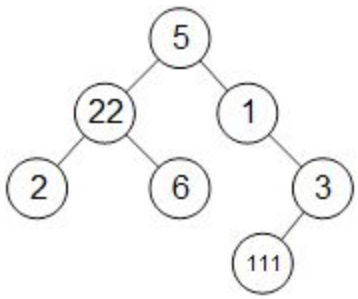
Стратегия за обхождане на дърво ще представяме чрез следния тип:

```
data Strategy = Inorder | Postorder | Preorder deriving (Show,Read)
```

Напишете функция, която обхожда дърво `t`, използвайки стратегия `s` и връща списък от неговите елементи. Елементите в списъка да са наредени точно в реда, в който са били открити при обхождането.

```
values :: Strategy -> (Tree a) -> [a]
values s t
```

Пример: нека е дадено следното дърво:



При `Inorder` обхождане трябва да се получи списъкът `[2, 22, 6, 5, 1, 111, 3]`.

Упътване: повече информация за работата с типове в Haskell можете да намерите тук: http://learnyouahaskell.com/making-our-own-types-and-typeclasses

Задача 2

Цвят ще представяме чрез следния тип:

```
data Rgb = Rgb { red   :: Word8
               , green :: Word8
               , blue  :: Word8 } deriving (Show,Read)
```

За да можете да работите с типа, трябва да включите `Data.Word`:

```
import Data.Word
```

Изображение ще представяме чрез неговите размери и списък от списъци от пикселите му. В този списък, всеки подсписък представя един ред на изображението. Пръв в списъка е най-горният ред, следван от втория и т.н. Всеки ред се представя като списък от цветовите стойности за всеки пиксел, започвайки от най-левия и продължавайки към най-десния.

```
data Image = Image { width   :: Int
                   , height  :: Int
                   , content :: [[Rgb]] } deriving (Show,Read)
```

По-долу е даден пример за представянето на изображение с размери 3x2, чиито пиксели са описани в следната таблица:

RGB 255 0 0	RGB 255 128 0	RGB 255 255 0
RGB 0 255 0	RGB 255 255 255	RGB 128 255 128

```
Image 3 2 [[Rgb 255 0 0, Rgb 155 128 0,   Rgb 255 255 0],
           [Rgb 0 255 0, Rgb 255 255 255, Rgb 128 255 128]]
```

Реализирайте следните функции:

А)

```
grayscale :: Image -> Image
grayscale img
```

Тази функция трябва да преобразува изображение до grayscale. Алгоритъмът трябва да бъде следният:

- всеки пиксел се обработва самостоятелно;
- новата му стойност се пресмята като 0.30*R + 0.59*G + 0.11*B.

Б)

```
edgeDetect :: Image -> Image
edgeDetect img
```

Тази функция трябва да извършва просто разпознаване на ръбове, прилагайки Собеловия оператор. Този оператор работи върху **grayscale**-нати изображения по следния начин:

- Всеки grayscale-нат пиксел се обработва самостоятелно
- За всеки пиксел разглеждаме матрица 3x3, образувана от него и съседните му 8 пиксела
- Тази матрица умножаваме скаларно веднъж по матрицата `[[1,0,-1],[2,0,-2],[1,0,-1]]` и веднъж по `[[1,2,1],[0,0,0],[-1,-2,-1]]` (транспонираната ѝ)
    - ВАЖНО: в уикипедия първата матрица понякога не се визуализира правилно (има 2 вместо -2)
- Нормираме резултатите от двете скаларни умножения
- Резултата clamp-ваме в интервала [0;255], тъй като може да е извън този интервал
- Полученото е grayscale цветът на съответния пиксел в резултатното изображение

Упътване:

- `Word8` не е подходящ тип за междинните изчисления - най-добре използвайте `Float`
- Скаларното умножение от по-горе е известно още като конволюция, а дадените матрици като кърнъли.
- Можете да откриете примерното изображение от статията в PPM формат тук, както и примерен резултат от извикването на `edgeDetect`.
- Пикселите по ръба на изображението имат по-малко от 8 съседа. Ваш е изборът на стратегия за тях (вижте секцията “Edge handling” в статията за кърнълите)
- Алгоритъмът е удобен за императивни езици с масиви с директен достъп до елементите им. Може да помислите как по-хитро да “комбинирате” даден пиксел или цял ред/стълб от пиксели със своите съседи :) Може да забележите, че едната конволюция е “по редове”, а другата “по стълбове”.

В)

```
floodFill :: Rgb -> Int -> Int -> Image -> Image
floodFill color x y img
```

Тази функция трябва да запълва област в изображение, използвайки Flood Fill, с помощта на BFS. Запълването трябва да стане с цвета `color`. Запълването да се осъществи върху пиксела с координати (`x`,`y`) и всички достижими от него, които са със същия цвят като него.

Г)

```
saveImage :: FilePath -> Image -> IO()
saveImage path img
```

Тази функция трябва да записва изображение в графичен файл в PPM формат.
Допустимо е към тази функция да не изготвяте Unit test-ове.

Д)

```
loadImage :: String -> IO Image
loadImage path
```

Тази функция трябва да зарежда изображение от графичен файл в PPM формат.
Допустимо е към тази функция да не изготвяте Unit test-ове.

Редакция от 01.01.2021 г. – Уточнение към точки Г) и Д): Форматът ще бъде P3 (текстов), няма да бъде P6 (двоичен). При отварянето на файла обаче, направете проверка, за да се уверите, че форматът е коректен; ако това не е така, може например да върнете празно изображение. Също, може да считате, че във файла няма да има коментари. Препоръчваме да се съобразите със следната препоръка относно формата: *"it's recommended that no line is longer than 76 characters"*. Може да не броите точно символите, а например през някакъв брой пиксели да извеждате нов ред, всеки пиксел да е на отделен ред и т.н.