Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет программной инженерии и компьютерной техники

Кафедра информатики и прикладной математики
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Функциональное программирование»

ЛАБОРАТОРНАЯ РАБОТА № 2

Студент: Григорьев Г. Г.

Преподаватель: Лаздин А.В.

Санкт-Петербург
2018

1. Определите класс типов Группа, https://ru.wikipedia.org/wiki/Группа_(математика).
2. Определите класс типов Кольцо (с единицей), https://ru.wikipedia.org/wiki/Кольцо_(математика).
3. Определите тип данных Кольцо вычетов по модулю n представителем класса типов Кольцо (с единицей), https://ru.wikipedia.org/wiki/Сравнение_по_модулю#Классы_вычетов.
4. Определите тип данных Двоичное дерево.
5. Сделайте Двоичное дерево представителем класса Foldable, реализовав обход дерева в глубину.
6. Сделайте Двоичное дерево представителем класса Foldable, реализовав обход дерева в ширину.
7. Расширение задачи 5. Реализуете три варианта обхода дерева: preorder, in-order, postorder, https://en.m.wikipedia.org/wiki/Tree_traversal.

Решение:

Ring.hs:

```
class Monoid g => Group g where
 inv :: g -> g


class Group a => Ring a where
 mul :: a -> a -> a
 unit :: a
 -- a * (b * c) === (a * b) * c
 --    one * a === a
 --    a * one === a
 -- a * (b + c) === a * b + a * c


data RingMod = RingMod { rmRem :: Int, rmMod :: Int } deriving (Show, Eq)


ringMod :: Int -> Int -> RingMod
ringMod a m = RingMod (a `mod` m) m
```

```
instance Monoid RingMod where
 mempty = ringMod 0 1
 mappend (RingMod a m) (RingMod b n) | m == n = ringMod (a + b) m
                                     | otherwise = undefined

instance Group RingMod where
 inv (RingMod a m) = ringMod (a - m) m

instance Ring RingMod where
 mul (RingMod a m) (RingMod b n) | m == n = ringMod (a * b) m
                                 | otherwise = undefined
 unit = ringMod 1 1
```

Tree.hs:

```
import Data.Foldable (toList)

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

instance Foldable Tree where
 foldMap _ Null = mempty
 foldMap f (Node x l r) = f x `mappend` preorder f l `mappend`
preorder f r

tree = Node 4 (Node 2 (Node 0 Null Null) (Node 6 Null Null)) (Node 7
(Node 8 Null Null) Null)

dfs :: Monoid m => (a -> m) -> Tree a -> m
dfs _ Null = mempty
```

```
dfs f (Node x l r) = f x `mappend` dfs f l `mappend` dfs f r


bfs :: Monoid m => (a -> m) -> Tree a -> m
bfs _ Null = mempty
bfs f t = bfs' mempty [t]
 where bfs' acc [] = acc
     bfs' acc (t:ts) = let acc' = acc `mappend` (f $ value t)
                in bfs' acc' $ ts ++ subtrees t
     subtrees t = [ sub | sub@(Node _ _ _) <- [left t, right t] ]


preorder :: Monoid m => (a -> m) -> Tree a -> m
preorder _ Null = mempty
preorder f (Node x l r) = f x `mappend` preorder f l `mappend`
preorder f r


inorder :: Monoid m => (a -> m) -> Tree a -> m
inorder _ Null = mempty
inorder f (Node x l r) = inorder f l `mappend` f x `mappend` inorder f
r


postorder :: Monoid m => (a -> m) -> Tree a -> m
postorder _ Null = mempty
postorder f (Node x l r) = postorder f l `mappend` postorder f r
`mappend` f x



perms :: [a] -> [[a]]
perms [] = []
perms [x] = [[x]]
perms xs = do let len = length xs
```

```
let flip (x:xs) = xs ++ [x]
rotation <- take len $ iterate flip xs
(y:ys)  <- return rotation
result  <- do tailPerm <- perms ys
              return $ y : tailPerm
return result
```

Санкт-Петербург
2018