

CS314. Функциональное программирование

Примеры решения задач на обработку списков

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

Содержание

- 1 Числа Фибоначчи
- 2 Ряды Коллатца
- 3 Подсчёт слов в строке
- 4 Определение вхождения подсписка в список
- 5 Шифр Цезаря
- 6 Анализ подходящих дробей для $\sqrt{2}$

Содержание

- 1 Числа Фибоначчи
- 2 Ряды Коллатца
- 3 Подсчёт слов в строке
- 4 Определение вхождения подсписка в список
- 5 Шифр Цезаря
- 6 Анализ подходящих дробей для $\sqrt{2}$

Числа Фибоначчи

Определение

Числами Фибоначчи называется числовая последовательность, задаваемая правилами:

$$F_0 = 0,$$

$$F_1 = 1,$$

$$F_n = F_{n-1} + F_{n-2}.$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Числа Фибоначчи

Определение

Числами Фибоначчи называется числовая последовательность, задаваемая правилами:

$$F_0 = 0,$$

$$F_1 = 1,$$

$$F_n = F_{n-1} + F_{n-2}.$$

0,	1,	1,	2,	3,	5,	8,	13,	21,	34,	55,	89,	144,...
	0,	1,	1,	2,	3,	5,	8,	13,	21,	34,	55,	...
	1,	1,	2,	3,	5,	8,	13,	21,	34,	55,	89,	...

Первая строка равна сумме второй и третьей!

Построение списка чисел Фибоначчи

```
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

```
ghci> take 13 fibs  
[0,1,1,2,3,5,8,13,21,34,55,89,144]
```

Содержание

- 1 Числа Фибоначчи
- 2 Ряды Коллатца**
- 3 Подсчёт слов в строке
- 4 Определение вхождения подсписка в список
- 5 Шифр Цезаря
- 6 Анализ подходящих дробей для $\sqrt{2}$

Пример: ряды Коллатца

Ряд Коллатца

- 1 Первое число — произвольное.
- 2 Следующее число — если предыдущее число чётное, то делим его на 2, в противном случае умножаем на 3 и прибавляем 1.

Пример

27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, ...

Пример: ряды Коллатца

Гипотеза Коллатца

С какого бы числа ни начинался ряд, в нём обязательно возникнет число 1.

Пример: ряды Коллатца

Гипотеза Коллатца

С какого бы числа ни начинался ряд, в нём обязательно возникнет число 1.

Задача

Длина скольких цепочек, начинающихся с чисел от 1 до 100, превосходит 15?

Алгоритм решения

- 1 Перебираем все стартовые числа.
- 2 Строим для каждого числа цепочки.
- 3 Отбираем все цепочки, длина которых больше 15.
- 4 Считаем количество оставшихся цепочек.

Пример: ряды Коллатца

Решение

```

chain :: Int -> [Int]
chain n = takeWhile (/=1) $ iterate next n
  where
    next n
      | even n = n `div` 2
      | odd  n = n * 3 + 1

longChains :: Int -> Int -> [[Int]]
longChains len lim = filter ((>len).length) $ map chain [1..lim]

answer :: Int
answer = length $ longChains 15 100

```

Содержание

- 1 Числа Фибоначчи
- 2 Ряды Коллатца
- 3 Подсчёт слов в строке**
- 4 Определение вхождения подсписка в список
- 5 Шифр Цезаря
- 6 Анализ подходящих дробей для $\sqrt{2}$

Подсчёт слов в строке

Задача

Дана строка. Подсчитать, сколько раз в ней встречается каждое слово.

Пример

```
ghci> wordNums ["ya ya ui ya"]  
[( "ya ",3),( "ui ",1)]
```

Тип функции

```
wordNums :: String -> [(String, Int)]
```

Компоненты решения

- Модуль `Data.List`.
- Разбиение на слова — функция `Data.List.words`.
- Сортировка слов — функция `Data.List.sort`.
- Группировка одинаковых слов — функция `Data.List.group`.
- Подсчёт слов в группах.

Используемые функции из модуля Data.List

```
ghci> :m +Data.List
ghci> words "всё это слова в этом предложении"
["всё", "это", "слова", "в", "этом", "предложении"]
ghci> words "всё  это слова в этом      предложении"
["всё", "это", "слова", "в", "этом", "предложении"]
ghci> sort [5,4,3,7,2,1]
[1,2,3,4,5,7]
ghci> sort ["бум", "бип", "бип", "бум", "бум"]
["бип", "бип", "бум", "бум", "бум"]
ghci> group [1,1,1,1,2,2,2,2,3,3,2,2,2,5,6,7]
[[1,1,1,1],[2,2,2,2],[3,3],[2,2,2],[5],[6],[7]]
```

Решение

```
import Data.List

wordNums :: String -> [(String, Int)]
wordNums = map (\(w:ws) -> (w, length ws + 1))
            . group . sort . words
```


Содержание

- 1 Числа Фибоначчи
- 2 Ряды Коллатца
- 3 Подсчёт слов в строке
- 4 Определение вхождения подсписка в список**
- 5 Шифр Цезаря
- 6 Анализ подходящих дробей для $\sqrt{2}$

Определение вхождения подсписка в список

Задача

Даны два списка. Определить, содержится ли первый список во втором.

Пример

```
ghci> "обед" `isIn` "победа"
```

```
True
```

```
ghci> [1,2] `isIn` [1,3,5]
```

```
False
```

Тип функции

```
isIn :: (Eq a) => [a] -> [a] -> Bool
```

Вспомогательная задача

Начинается ли один список с другого?

```
startsWith :: (Eq a) => [a] -> [a] -> Bool
xs `startsWith` ys = (and $ zipWith (==) xs ys)
                    && (length xs >= length ys)
```

```
ghci> "гавайи джо" `startsWith` "гавайи"
True
ghci> [1,2] `startsWith` [1,2,3]
False
ghci> "ха" `startsWith` "ха"
True
```

Функция Data.List.tails

```
ghci> tails "победа"  
["победа", "обеда", "беда", "еда", "да", "а", ""]  
ghci> tails [1,2,3]  
[[1,2,3],[2,3],[3],[]]
```

Функция Data.List.any

```
ghci> any (>4) [1,2,3]
```

```
False
```

```
ghci> any (=='H') "Gregory House"
```

```
True
```

```
ghci> any (\x -> 5 < x && x < 10) [1,4,11]
```

```
False
```

Решение

```
import Data.List
```

```
isIn :: (Eq a) => [a] -> [a] -> Bool
```

```
needle `isIn` haystack = any (`startsWith` needle) (tails haystack)
```

Аналогичные стандартные функции из модуля Data.List

- isPrefixOf
- isInfixOf
- isSuffixOf

Содержание

- 1 Числа Фибоначчи
- 2 Ряды Коллатца
- 3 Подсчёт слов в строке
- 4 Определение вхождения подсписка в список
- 5 Шифр Цезаря**
- 6 Анализ подходящих дробей для $\sqrt{2}$

Шифр Цезаря

Задача

Реализовать кодирование и декодирование сообщения сдвигом каждого символа на заданное число позиций.

`encode :: Int -> String -> String`

`decode :: Int -> String -> String`

Работа с символами

```
ghci> :m +Data.Char
ghci> ord 'a'
97
ghci> chr 97
'a'
ghci> map ord "abcdefgh"
[97,98,99,100,101,102,103,104]
```

Решение

```
import Data.Char
```

```
encode :: Int -> String -> String
```

```
encode offset msg = map (chr . (+offset) . ord) msg
```

```
decode :: Int -> String -> String
```

```
decode shift msg = encode (negate shift) msg
```

```
ghci> encode 3 "hello"  
"khood"
```

```
ghci> decode 3 "khood"  
"hello"
```

Содержание

- 1 Числа Фибоначчи
- 2 Ряды Коллатца
- 3 Подсчёт слов в строке
- 4 Определение вхождения подсписка в список
- 5 Шифр Цезаря
- 6 Анализ подходящих дробей для $\sqrt{2}$**

Основные понятия

Цепная дробь для $\sqrt{2}$

$$\sqrt{2} = 1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \dots}}}} = 1.414213\dots$$

Основные понятия

Подходящие дроби

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$

- $1 + \frac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\dots$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\dots$

Постановка задачи

Среди первых восьми подходящих дробей для $\sqrt{2}$

$$\frac{3}{2}, \frac{7}{5}, \frac{17}{12}, \frac{41}{29}, \frac{99}{70}, \frac{239}{169}, \frac{577}{408}, \frac{1393}{985}$$

только в одной количество знаков в числителе превосходит количество знаков в знаменателе. Сколько таких дробей среди первой тысячи подходящих дробей для $\sqrt{2}$?

Компоненты решения

- Представление для дробей с операциями — модуль `Data.Ratio`.
- Список подходящих дробей.
- Предикат для дроби: числитель длиннее знаменателя.
- Фильтр по предикату и подсчёт количества оставшихся дробей.

Модуль Data.Ratio

Дроби и операции с ними

```
ghci> :m +Data.Ratio
```

```
ghci> 1 % 4
```

```
1 % 4
```

```
ghci> 1 % 2 + 1 % 2
```

```
1 % 1
```

```
ghci> (2%3) * (3%5)
```

```
2 % 5
```

```
ghci> (2%3) / (3%5)
```

```
10 % 9
```

```
ghci> numerator (1%2)
```

```
1
```

```
ghci> denominator (1%2)
```

```
2
```

Предикат для дробей

```
longer :: Integer -> Integer -> Bool  
n `longer` m = length (show n) > length (show m)
```

```
predicate :: Rational -> Bool  
predicate ratio = numerator ratio `longer` denominator ratio
```

Вычисление ответа

```
expansions :: [Rational]  
expansions = undefined
```

```
answer :: Int  
answer = length $ filter predicate $ take 1000 expansions
```

Список подходящих дробей

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$

- $1 + \frac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\dots$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\dots$

Список подходящих дробей

- $1 + \boxed{\frac{1}{2}} = \frac{3}{2} = 1.5$

- $1 + \frac{1}{2 + \boxed{\frac{1}{2}}} = \frac{7}{5} = 1.4$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\dots$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\dots$

Список подходящих дробей

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$

- $1 + \frac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\dots$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\dots$

Список подходящих дробей

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$

- $1 + \frac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\dots$

- $1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\dots$

Построение списка подходящих дробей

- Строим числовую последовательность (функция `iterate`):

$$r_0 = \frac{1}{2},$$

$$r_n = \frac{1}{2+r_{n-1}}.$$

- Увеличиваем каждый элемент на 1: $e_n = 1 + r_n$.

Список подходящих дробей

`expansions :: [Rational]`

`expansions = map (+1) $ iterate (\r -> 1 / (2 + r)) (1%2)`

Полное решение

```
import Data.Ratio
```

```
expansions :: [Rational]
```

```
expansions = map (+1) $ iterate (\r -> 1 / (2 + r)) (1%2)
```

```
longer :: Integer -> Integer -> Bool
```

```
longer n1 n2 = length (show n1) > length (show n2)
```

```
predicate :: Rational -> Bool
```

```
predicate ratio = numerator ratio `longer` denominator ratio
```

```
answer :: Int
```

```
answer = length $ filter predicate $ take 1000 expansions
```