CS212. Парадигмы и технологии программирования: часть 1, функциональное программирование

Примеры 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

Числа Фибоначчи

Числа Фибоначчи

Определение

Числами Фибоначчи называется числовая последовательность, задаваемая правилами:

$$F_0 = 0,$$
$$F_1 = 1,$$
$$F_n = F_{n-1} + F_{n-2}.$$

$0, \quad 1, \quad 1, \quad 2, \quad 3, \quad 5, \quad 8, \quad 13, \quad 21, \quad 34, \quad 55, \quad 89, \quad 144, \ldots$

Числа Фибоначчи

Определение

Числами Фибоначчи называется числовая последовательность, задаваемая правилами:

$$F_0 = 0,$$
$$F_1 = 1,$$
$$F_n = F_{n-1} + F_{n-2}.$$

$$\begin{array}{ccccccccccccc}
0, & 1, & 1, & 2, & 3, & 5, & 8, & 13, & 21, & 34, & 55, & 89, & 144, \ldots \\
 & & 0, & 1, & 1, & 2, & 3, & 5, & 8, & 13, & 21, & 34, & 55, \ldots \\
 & & 1, & 1, & 2, & 3, & 5, & 8, & 13, & 21, & 34, & 55, & 89, \ldots
\end{array}$$

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]
```

- «Завязывание узлов»: `fibs` зависит от `fibs`

Ряды Коллатца

Ряды Коллатца

Определение

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

Ряды Коллатца

Определение

1. Первое число — произвольное.
2. Следующее число — если предыдущее число 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, …

Ряды Коллатца: задача

Гипотеза Коллатца

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

Ряды Коллатца: задача

Гипотеза Коллатца

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

Задача

Длина скольких цепочек, начинающихся с чисел от 1 до 100, превосходит 15 (без учёта последней единицы)?

Ряды Коллатца: задача

Гипотеза Коллатца

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

Задача

Длина скольких цепочек, начинающихся с чисел от 1 до 100, превосходит 15 (без учёта последней единицы)?

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.

Ряды Коллатца: решение

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

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

answer :: Int
answer = length $ longChains 15 100
```

Подсчёт слов в строке

Подсчёт слов в строке

Задача

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

Пример

```
ghci> wordNums "ho hey ho ho"
[("hey",1),("ho",3)]
```

Тип функции

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

Компоненты решения

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

Используемые функции из модуля Data.List

```
ghci> import 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]]
```

```
group :: Eq a => [a] -> [[a]]
```

Решение

```haskell
import Data.List

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

- as-паттерн `ws@(w:_)`, здесь `ws` — это заданный список целиком, а `w` — это его голова

Определение вхождения подсписка в список

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?

```haskell
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> "xa" `startsWith` "xa"
True
```

Функции Data.List.tails и any

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

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

```haskell
import Data.List

isIn :: (Eq a) => [a] -> [a] -> Bool
isIn needle haystack =
    any (`startsWith` needle) (tails haystack)
```

Решение

```haskell
import Data.List

isIn :: (Eq a) => [a] -> [a] -> Bool
isIn needle haystack =
    any (`startsWith` needle) (tails haystack)
```

Аналогичные функции из модуля Data.List

- `isPrefixOf`
- `isInfixOf`
- `isSuffixOf`

Шифр Цезаря

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

Решение

```haskell
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"
"khoor"
ghci> decode 3 "khoor"
"hello"
```

Анализ подходящих дробей для $\sqrt{2}$

Цепная дробь для $\sqrt{2}$

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cdots}}}} = 1.414213\ldots$$

Подходящие дроби

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$
- $1 + \cfrac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\ldots$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\ldots$

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` и тип `Rational`.
- 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> import 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
longer n m = length (show n) > length (show m)

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

Вычисление ответа

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

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

Список подходящих дробей

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$
- $1 + \cfrac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\ldots$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\ldots$

Список подходящих дробей

- $1 + \boxed{\frac{1}{2}} = \frac{3}{2} = 1.5$
- $1 + \cfrac{1}{2 + \boxed{\frac{1}{2}}} = \frac{7}{5} = 1.4$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\ldots$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\ldots$

Список подходящих дробей

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$
- $1 + \cfrac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\ldots$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\ldots$

Список подходящих дробей

- $1 + \frac{1}{2} = \frac{3}{2} = 1.5$
- $1 + \cfrac{1}{2 + \frac{1}{2}} = \frac{7}{5} = 1.4$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}} = \frac{17}{12} = 1.41666\ldots$
- $1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{2}}}} = \frac{41}{29} = 1.41379\ldots$

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: $e_n = 1 + r_n$.

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

```haskell
import Data.Ratio

expansions :: [Rational]
expansions = map (+1)
             $ iterate (\r -> 1 / (2 + r)) (1%2)

answer = length $ filter predicate
                $ take 1000 expansions
  where
    longer n m = length (show n) > length (show m)
    predicate ratio = longer (numerator ratio)
                             (denominator ratio)
```

Полное решение

```haskell
import Data.Ratio

expansions :: [Rational]
expansions = map (+1)
             $ iterate (\r -> 1 / (2 + r)) (1%2)

answer = length $ filter predicate
                $ take 1000 expansions
  where
    longer n m = length (show n) > length (show m)
    predicate ratio = longer (numerator ratio)
                             (denominator ratio)
```

```
ghci> answer
153
```