CS314. Функциональное программирование

Лекция 3. Обработка списков

В. Н. Брагилевский

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

9 сентября 2016 г.

Списки в языке Haskell

Фрагмент исходного кода (lists.hs)

```
xs = [1,3,9]
ys = 5 : xs
zs = xs ++ ys
```

Сессия ghci

```
ghci> :load lists
ghci> xs
[1,3,9]
ghci> ys
[5,1,3,9]
ghci> zs
[1,3,9,5,1,3,9]
```

Содержание

Доступ к элементам и анализ списков

```
head :: [a] -> a        -- голова списка
tail :: [a] -> [a]      -- хвост списка
length :: [a] -> Int    -- длина списка
last :: [a] -> a        -- последний элемент списка
init :: [a] -> [a]      -- начальная часть списка
null :: [a] -> Bool     -- пуст ли список?
elem :: Eq a => a -> [a] -> Bool  -- содержится ли элемент
                                  -- в списке?

take, drop :: Int -> [a] -> [a]
splitAt :: Int -> [a] -> ([a], [a])

sum, product :: Num a => [a] -> a
maximum, minimum :: Ord a => [a] -> a

and, or :: [Bool] -> Bool
```

Формирование списков

```
reverse :: [a] -> [a]   -- обращение списка
concat :: [[a]] -> [a] -- соединение списков в один
repeat :: a -> [a]      -- создание бесконечного списка копий
replicate :: Int -> a -> [a] -- реплицирование элемента
cycle :: [a] -> [a]     -- бесконечное повторение списка
zip :: [a] -> [b] -> [(a, b)] -- спаривание элементов списков
unzip :: [(a, b)] -> ([a], [b]) -- разъединение пар
```

Функции на списках символов (строках)

```
lines :: String -> [String] -- разделение на строки (по \n)
words :: String -> [String] -- разбиение на слова
                            -- (по пробелам)

unlines :: [String] -> String
unwords :: [String] -> String
```

Содержание

Функция map: определение и примеры

```
map :: (a -> b) -> [a] -> [b]
map _ [ ] = [ ]
map f (x:xs) = f x : map f xs
```

```
ghci> map (+3) [1,5,3,1,6]
[4,8,6,4,9]
ghci> map (++ "!") ["БУХ", "БАХ", "ПАФ"]
["БУХ!","БАХ!","ПАФ!"]
ghci> map fst [(1,2),(3,5),(6,3),(2,6),(2,5)]
[1,3,6,2,2]
ghci> map (map (^2)) [[1,2],[3,4,5,6],[7,8]]
[[1,4],[9,16,25,36],[49,64]]
```

Функция filter: определение и примеры

```
filter :: (a -> Bool) -> [a] -> [a]
filter _ [ ] = [ ]
filter p (x:xs)
   | p x = x : filter p xs
   | otherwise = filter p xs
```

```
ghci> filter (>3) [1,5,3,2,1,6,4,3,2,1]
[5,6,4]
ghci> filter (\x -> x `mod` 3 == 0) [1..10]
[3,6,9]
ghci> filter (<15) $ filter even [1..20]
[2,4,6,8,10,12,14]
```

Пример: «быстрая» сортировка

```
quicksort :: (Ord a) => [a] -> [a]
quicksort [ ] = [ ]
quicksort (x:xs) = quicksort (filter (<= x) xs)
                   ++ [x] ++ quicksort (filter (> x) xs)
```

Пример: сумма квадратов

Задача

Найти сумму квадратов чисел из диапазона от 1 до 100, делящихся на 3 или 5.

Алгоритм решения

- Формируем список чисел.
- Оставляем числа, делящиеся на 3 или 5.
- Возводим каждое число в квадрат.
- Вычисляем сумму.

```
ghci> sum $ map (^2)
        $ filter (\n -> (n `mod` 3) * (n `mod` 5) == 0)
                 [1..100]
164036
```

Генераторы списков как аналог map и filter

```
ghci> sum $ map (^2)
          $ filter (\n -> (n `mod` 3) * (n `mod` 5) == 0)
                   [1..100]
164036
```

```
ghci> sum [n^2 | n <- [1..100],
                 (n `mod` 3) * (n `mod` 5) == 0]
164036
```

Пример: поиск числа

Задача

Найти наибольшее число, меньшее 100000, которое делится на 3829.

Алгоритм решения

- Формируем список кандидатов (в порядке убывания).
- Фильтруем список, оставляя только те, которые делятся на 3829.
- Берём первый элемент получившегося списка (head).

Решение

```
largestDiv :: Integer -> Integer -> Integer
largestDiv d l = head $ filter (\x -> x `mod` d == 0)
                               [l,l-1..]

ghci> largestDiv 3829 100000
99554
```

Пример: поиск числа

Версия решения с использованием сечений и композиции

```
largestDiv d l = head $ filter ((==0).(`mod` d)) [l,l-1..]
```

И ещё одна версия

```
largestDiv d = head . filter ((==0).(`mod` d))
                    . iterate (subtract 1)
```

```
ghci> :t iterate
iterate :: (a -> a) -> a -> [a]
ghci> take 6 $ iterate (^2) 2
[2,4,16,256,65536,4294967296]
ghci> take 7 $ iterate ('x':) ""
["","x","xx","xxx","xxxx","xxxxx", "xxxxxx"]
```

Вычисление квадратного корня с помощью списков

```
sqrt' y = head $ filter goodEnough guesses
  where
    goodEnough x = abs (sqr x - y) < eps
    improve x = average x (y/x)
    guesses = iterate improve 1
```

Функции takeWhile и dropWhile

Определение takeWhile

```
takeWhile :: (a -> Bool) -> [a] -> [a]
takeWhile _ [ ] = [ ]
takeWhile p (x:xs)
    | p x = x : takeWhile p xs
    | otherwise = [ ]
```

Определение dropWhile

```
dropWhile :: (a -> Bool) -> [a] -> [a]
dropWhile _ [ ] = [ ]
dropWhile p (x:xs)
    | p x = dropWhile p xs
    | otherwise = xs
```

Функции takeWhile и dropWhile: примеры

```
ghci> takeWhile (<10) [2,4..]
[2,4,6,8]
ghci> dropWhile (<10) [2,4..20]
[10,12,14,16,18,20]
ghci> takeWhile (/=' ') "hello world"
"hello"
ghci> dropWhile (`elem` ['a'..'z']) "hello world"
" world"
```

Функция zipWith

Определение

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith _ [ ] _ = [ ]
zipWith _ _ [ ] = [ ]
zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys
```

Функция zipWith

```
ghci> zipWith (+) [4,2,5,6] [2,6,2,3]
[6,8,7,9]
ghci> zipWith max [6,3,2,1] [7,3,1,5]
[7,3,2,5]
ghci> zipWith (*) [2,2,2,2,2] [1..]
[2,4,6,8,10]
ghci> zipWith (++) ["Hypno ", "Flareon "] ["Psychic", "Fire"]
["Hypno Psychic","Flareon Fire"]
ghci> zipWith (zipWith (*)) [[1,2,3],[3,5,6],[2,3,4]]
                            [[3,2,2],[3,4,5],[5,4,3]]
[[3,4,6],[9,20,30],[10,12,12]]
```

Функция zipWith

Применение

```
zip' :: [a] -> [b] -> [(a,b)]
zip' = zipWith (,)

sup :: Ord a => [a] -> [a] -> [a]
sup = zipWith max
```

Примеры

```
ghci> zip' [1,2,3] [2,4..]
[(1,2),(2,4),(3,6)]
ghci> sup [10, 2, 5] [3, 5, 7]
[10,5,7]
```

Некоторые функции обработки списков

```
??? :: (a -> Bool) -> [a] -> Bool
??? :: (a -> Bool) -> [a] -> ([a], [a])
??? :: (a -> Bool) -> [a] -> [Int]
??? :: (a -> a -> Bool) -> [a] -> [a]
??? :: (a -> a -> Bool) -> [a] -> [[a]]
??? :: (a -> b -> a) -> a -> [b] -> a
??? :: (a -> b -> a) -> a -> [b] -> [a]
```

Hoogλe — поиск по функциям (http://www.haskell.org/hoogle/)

Примеры поисковых запросов:

- map
- (a -> b) -> [a] -> [b]

Содержание

Pascal

```
s := 0;
for i:= 1 to 10 do
  s := s + arr[i];
```

C#

```
s = 0;
foreach (n in numbers)
  s += n;
```

Основные компоненты кода

- Проход по структуре данных или диапазону (цикл).
- Аккумулирующая переменная.
- Текущее значение.
- Вычисление в теле цикла.

Содержание

Определение

Свёртки — это семейство ФВП, обрабатывающих все компоненты рекурсивной структуры данных и вычисляющих в результате некоторое значение (аккумулятор). Обычно свёртка задаётся комбинирующей функцией, структурой данных и, возможно, начальным значением аккумулятора.

```
foldl :: (b -> a -> b) -> b -> [a] -> b
foldl _ z [ ]     = z
foldl f z (x:xs) = foldl f (f z x) xs

foldr :: (a -> b -> b) -> b -> [a] -> b
foldr _ z [ ]     = z
foldr f z (x:xs) = f x (foldr f z xs)
```

Порядок сворачивания списка

Левая свёртка

```
foldl g z [3,4,5,6] === g (g (g (g z 3) 4) 5) 6
                    === (((z `g` 3) `g` 4) `g` 5) `g` 6
```

Правая свёртка

```
foldr f z [3,4,5,6] === f 3 (f 4 (f 5 (f 6 z)))
                    === 3 `f` (4 `f` (5 `f` (6 `f` z)))
```

Простейшие примеры

```
sum' :: (Num a) => [a] -> a
sum' xs = foldl (+) 0 xs

sum'' :: (Num a) => [a] -> a
sum'' = foldl (+) 0

product' :: (Num a) => [a] -> a
product' = foldl (*) 1

elem' :: (Eq a) => a -> [a] -> Bool
elem' y ys = foldl (\acc x -> if x == y then True else acc)
                   False
                   ys
```

Примеры

Количество положительных элементов списка

```
countPositive = foldl (\c x -> c + if x > 0 then 1 else 0) 0
```

Сумма и произведение элементов списка

```
sumprod = foldl (\(s,p) x -> (s+x, p*x)) (0, 1)
```

Построение списков свёртками

```haskell
map' :: (a -> b) -> [a] -> [b]
map' f xs = foldr (\x acc -> f x : acc) [] xs

map'' :: (a -> b) -> [a] -> [b]
map'' f xs = foldr ((:).f) [] xs

filter' :: (a -> Bool) -> [a] -> [a]
filter' p = foldr (\x acc -> if p x then x : acc else acc) []

reverse' :: [a] -> [a]
reverse' = foldl (\acc x -> x : acc) []

reverse'' :: [a] -> [a]
reverse'' = foldl (flip (:)) []
```

Функции foldl1 и foldr1

Определение

```
foldl1 :: (a -> a -> a) -> [a] -> a
foldl1 f (x:xs) =  foldl f x xs
foldl1 _ [ ] =  error "foldl1: empty list"

foldr1 :: (a -> a -> a) -> [a] -> a
foldr1 _ [x] =  x
foldr1 f (x:xs) =  f x (foldr1 f xs)
foldr1 _ [ ] =  error "foldr1: empty list"
```

Простейшие примеры

```
maximum' :: (Ord a) => [a] -> a
maximum' = foldl1 max

last' :: [a] -> a
last' = foldl1 (\ _ x -> x)
```

Как запомнить?

- «Северный ветер дует с севера»
- `http://foldl.com`
- `http://foldr.com`

Содержание

Сканирование списка

```
ghci> :t scanl
scanl :: (b -> a -> b) -> b -> [a] -> [b]
ghci> scanl (+) 0 [3,5,2,1]
[0,3,8,10,11]
ghci> :t scanr
scanr :: (a -> b -> b) -> b -> [a] -> [b]
ghci> scanr (+) 0 [3,5,2,1]
[11,8,3,1,0]
ghci> scanl1 (\acc x -> if x > acc then x else acc)
             [3,4,5,3,7,9,2,1]
[3,4,5,5,7,9,9,9]
ghci> scanl (flip (:)) [ ] [3,2,1]
[[ ],[3],[2,3],[1,2,3]]
```

- Сканирование сохраняет промежуточные результаты свёртки.

Пример

Задача

Определить наибольшее количество последовательных натуральных чисел, сумма квадратных корней которых не превосходит 1000.

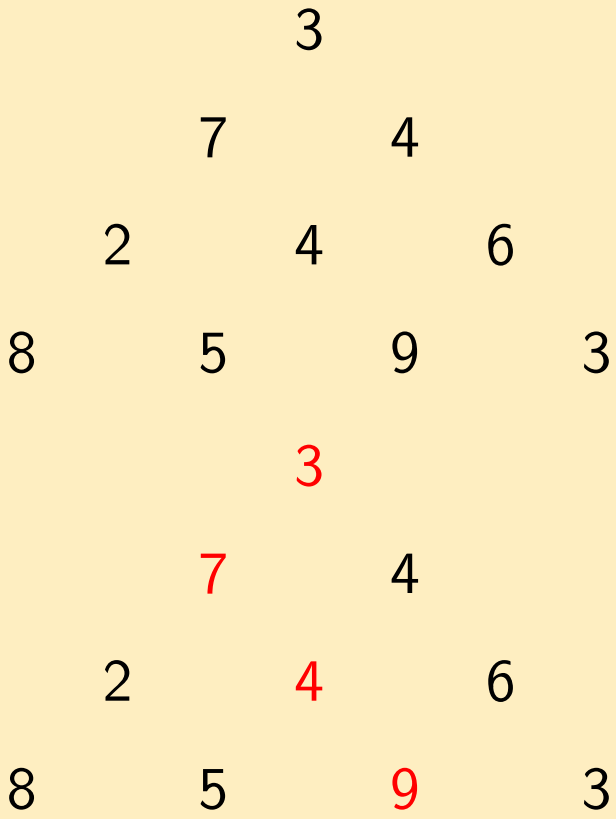
```
answer = length $ takeWhile (<= 1000)
                $ scanl1 (+) $ map sqrt [1..]
```

```
ghci> answer
130
ghci> sum (map sqrt [1..130])
993.6486803921487
ghci> sum (map sqrt [1..131])
1005.0942035344083
```

Содержание

Постановка задачи

Имеется числовой треугольник:



Необходимо вычислить максимальную сумму для пути от вершины к основанию. Допустим спуск либо влево, либо вправо.

Идея решения

$$\begin{array}{ccccccc} & & 7+3 & & 4+3 & & \\ & 2 & & 4 & & 6 & \\ 8 & & 5 & & 9 & & 3 \end{array}$$

$$\begin{array}{ccccccc} & & 10 & & 7 & & \\ & 2 & & 4 & & 6 & \\ 8 & & 5 & & 9 & & 3 \end{array}$$

$$\begin{array}{ccccccc} & & 2+10 & & 4+10 & & 6+7 \\ 8 & & & 5 & & 9 & & 3 \end{array}$$

$$\begin{array}{ccccc} 12 & & 14 & & 13 \end{array}$$

Компоненты решения

- Каждый уровень — список чисел.
- Треугольник в целом — список уровней.
- Сворачиваем все уровни в один уровень с максимальными путями.
- Находим максимум.

Переход от одного уровня к другому

```
      12      14      13
   8      5      9      3
```

```
          12                    14                    13
   8+12          5+max{12,14}          9+max{14,13}          3+13
          12                    14                    13
   8+max{0,12}   5+max{12,14}          9+max{14,13}          3+max{13,0}
```

```
      0      12      14      13
      12     14      13      0
-------------------------------
max
      12     14      14      13
```

Переход от одного уровня к другому

```
downstep :: [Int] -> [Int] -> [Int]
downstep upper lower = zipWith (+) lower
                     $ zipWith max (0:upper) (upper ++ [0])
```

Вычисление ответа

```
answer :: [[Int]] -> Int
answer = maximum . foldl1 downstep
```

Полное решение

```
downstep :: [Int] -> [Int] -> [Int]
downstep upper lower = zipWith (+) lower
                       $ zipWith max (0:upper) (upper ++ [0])
answer :: [[Int]] -> Int
answer = maximum . foldl1 downstep
```

```
ghci> answer [[3],[7,4],[2,4,6],[8,5,9,3]]
23
```