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

Лекция 2. Обработка списков и функции высших порядков

В. Н. Брагилевский

14 февраля 2018 г.

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

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]
```

Простейшие функции для обработки списков

Доступ к элементам списка

```haskell
head :: [a] -> a        -- голова списка
tail :: [a] -> [a]      -- хвост списка
last :: [a] -> a        -- последний элемент списка
init :: [a] -> [a]      -- список без последнего
                        -- элемента

-- Первые n элементов
take :: Int -> [a] -> [a]
-- Все элементы кроме n первых
drop :: Int -> [a] -> [a]

splitAt :: Int -> [a] -> ([a], [a])
```

Простейший анализ содержимого списков

```haskell
length :: Foldable t => t a -> Int
-- пуст ли список?
null :: Foldable t => t a -> Bool
-- содержится ли элемент в списке?
elem :: (Eq a, Foldable t) => a -> t a -> Bool

sum, product :: (Num a, Foldable t) => t a -> a

maximum :: (Ord a, Foldable t) => t a -> a
minimum :: (Ord a, Foldable t) => t a -> a

and, or ::  Foldable t => t Bool -> Bool
```

Формирование списков

```haskell
reverse :: [a] -> [a]  -- обращение списка
-- соединение списков в один
concat :: Foldable t => t [a] -> [a]
-- бесконечное повторение элемента
repeat :: a -> [a]
-- n-кратное повторение элемента
replicate :: Int -> a -> [a]
-- бесконечное повторение списка
cycle :: [a] -> [a]
-- спаривание элементов списков
zip :: [a] -> [b] -> [(a, b)]
-- разъединение пар
unzip :: [(a, b)] -> ([a], [b])
```

Функции на списках символов (строках)

```haskell
-- разделение на строки (по \n)
lines :: String -> [String]

--разбиение на слова (по пробелам)
words :: String -> [String]

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

ФВП для работы со списками

Функция map: определение и примеры

```haskell
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

```haskell
quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (x:xs) = lower ++ [x] ++ upper
  where
    lower = quicksort $ filter (<= x) xs
    upper = 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

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

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

Нотация для формирования списков

map и filter

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

Генераторы списков (list comprehensions)

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

$$\{n^2 \mid n \in [1..100], n \text{ делится на } 3 \text{ или } 5\}$$

Пример: поиск числа

Задача

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

Алгоритм решения

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

Пример: поиск числа

Решение

```
largestDivisible :: Integer -> Integer -> Integer
largestDivisible divisor lim =
     head $ filter (\x -> mod x divisor == 0)
                   [lim, lim-1..]

ghci> largestDivisible 3829 100000
99554
```

Пример: поиск числа

Версия решения с использованием сечений и композиции

```
largestDivisible divisor lim =
    head $ filter ((==0).(`mod` divisor))
                  [lim,lim-1..]
```

И ещё одна версия

```
largestDivisible divisor =
    head . filter ((==0).(`mod` divisor))
         . iterate (subtract 1)
```

Функция 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"]
```

Метод Ньютона на списках

```haskell
newton improve check eps y =
    head $ filter goodEnough guesses
  where
    goodEnough x = abs (check x - y) < eps
    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"
```

Сравнение filter и takeWhile на бесконечных списках

```
ghci> takeWhile (<10) [2,4..]
[2,4,6,8]
ghci> filter (<10) [2,4..]
????

ghci> filter (<10) [2,4..]
[2,4,6,8
```

Функция zipWith

Определение

```haskell
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

```haskell
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

URL: `http://www.haskell.org/hoogle/`

Примеры поисковых запросов:

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

Свёртки

Pascal

```pascal
(* arr - массив*)
s := 0;
for i:= 1 to 10 do
  s := s + arr[i];
```

C#

```csharp
// numbers - IEnumerable
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.

Свёртки

Левая и правая свёртки

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

foldr :: Foldable t => (a -> b -> b) -> b -> t 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
```

Правая свёртка

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

Простейшие примеры

```haskell
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)
```

Среднее арифметическое элементов списка

```
mean xs = sum / len
  where
    step (s, l) x = (s+x, l+1)
    (sum, len) = foldl step (0, 0) xs
```

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 step []
  where
    step x acc
      | p x = x : acc
      | otherwise = acc
```

Построение списков свёртками: reverse

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

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

Функции foldl1 и foldr1

Определение

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

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

Простейшие примеры

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

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

Свёртка бесконечных списков

Функция and'

```
and' :: [Bool] -> Bool
and' xs = foldr (&&) True xs
```

Порядок вычисления

```
and' [True,False,True]
    === True && (False && (True && True))
```

Случай бесконечного списка

```
and' (repeat False)
     === False && (False && (False && (False ...
```

Как запомнить?

- «Северный ветер дует с севера»

```
foldl :: (b -> a -> b) -> b -> [a] -> b
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Свёртки

Сканирование списка

Сканирование списка

```
foldl :: Foldable t =>
         (b -> a -> b) -> b -> t a -> b
scanl :: (b -> a -> b) -> b -> [a] -> [b]

foldr :: Foldable t =>
         (a -> b -> b) -> b -> t a -> b
scanr :: (a -> b -> b) -> b -> [a] -> [b]
```

```
ghci> scanl (+) 0 [3,5,2,1]
[0,3,8,10,11]
ghci> scanr (+) 0 [3,5,2,1]
[11,8,3,1,0]
ghci> scanl (flip (:)) [] [3,2,1]
[[],[3],[2,3],[1,2,3]]
```

- Сканирование — это свёртка с сохранением промежуточных значений.
- Имеются версии `scanl1` и `scanr1` без начального значения:

```
scanl1 :: (a -> a -> a) -> [a] -> [a]
scanr1 :: (a -> a -> a) -> [a] -> [a]
```

Пример

Задача

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

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

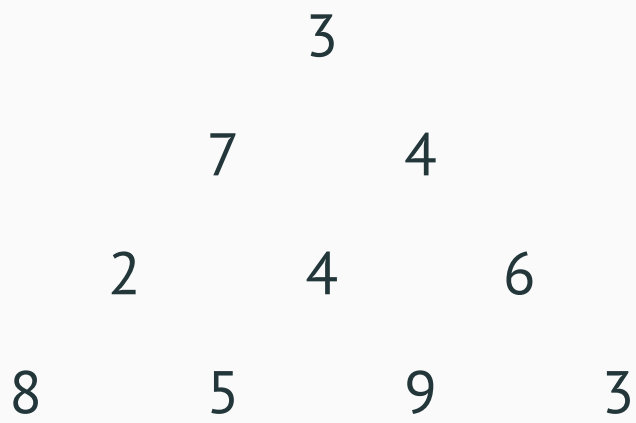
```
ghci> answer
130
ghci> sum (map sqrt [1..130])
993.6486803921487
ghci> sum (map sqrt [1..131])
1005.0942035344083
```

Свёртки

Пример: путь в треугольнике

Постановка задачи

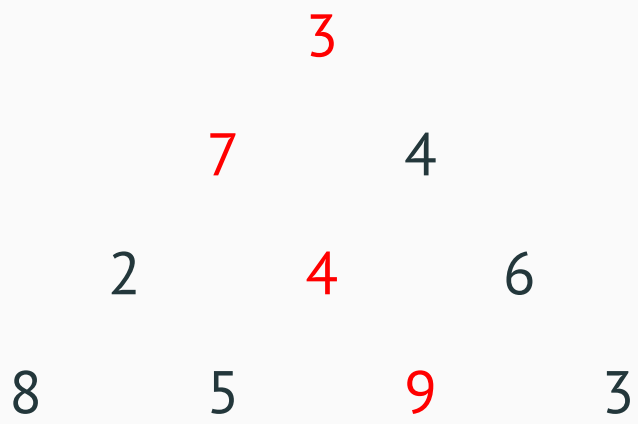
Имеется числовой треугольник:



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

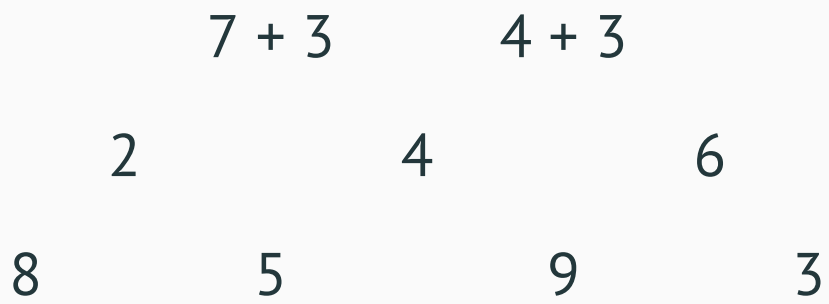
Постановка задачи

Имеется числовой треугольник:

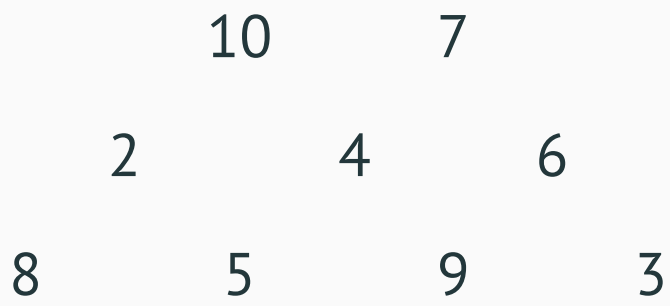


Необходимо вычислить максимальную сумму для пути от вершины к основанию. Допустим спуск либо влево, либо вправо.

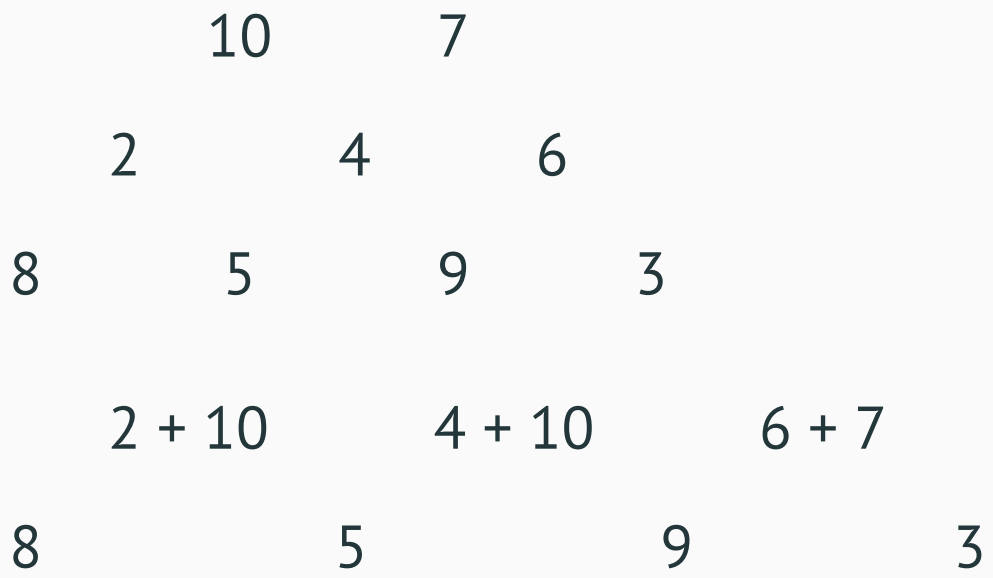
Идея решения



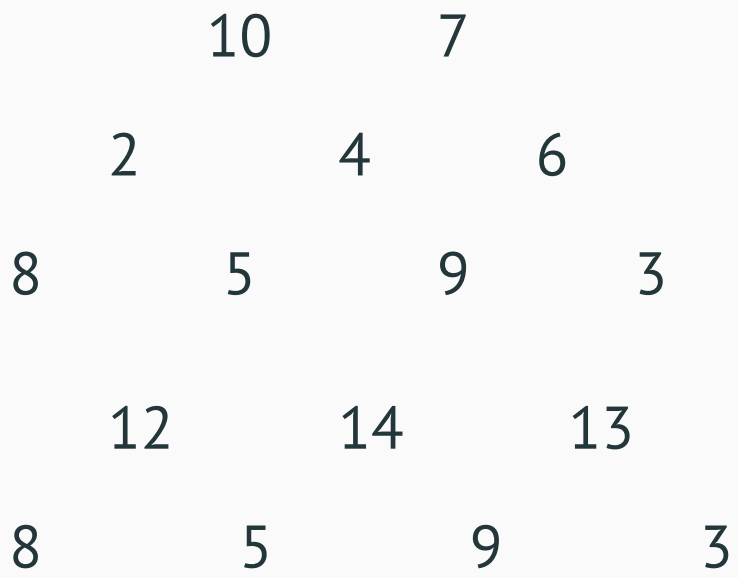
Идея решения



Идея решения

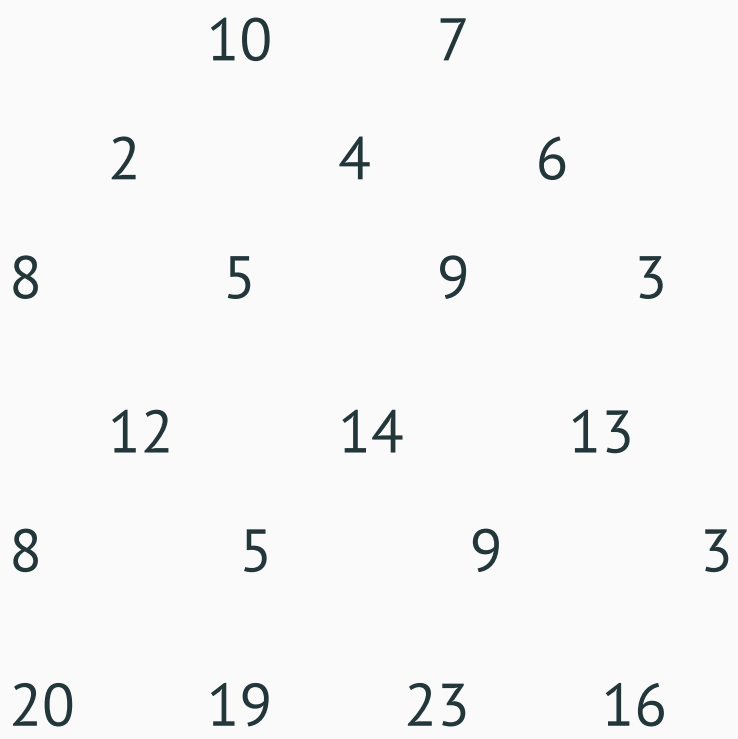


Идея решения



Идея решения

Идея решения



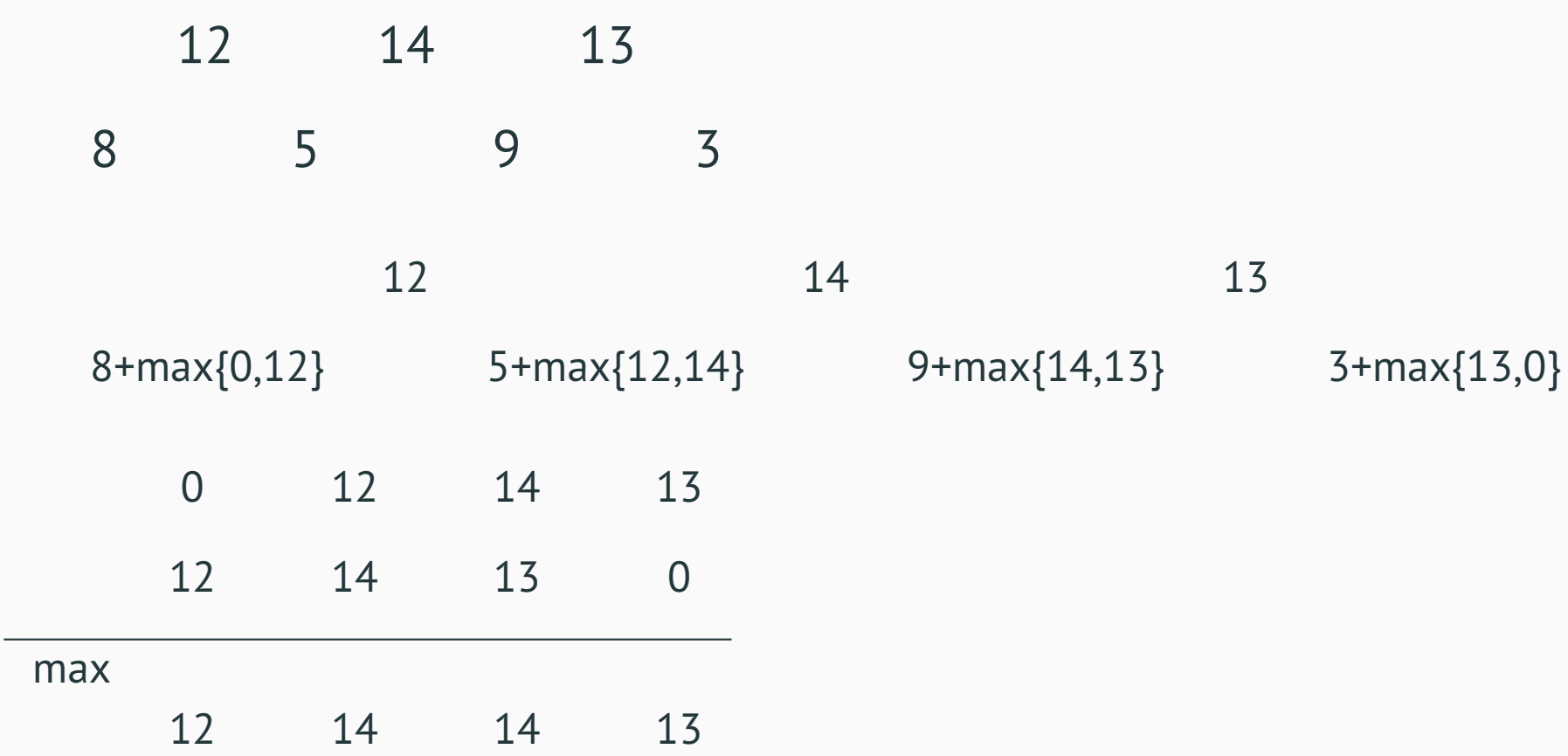
Компоненты решения

- 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

Переход от одного уровня к другому

Переход от одного уровня к другому



Переход от одного уровня к другому

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

Вычисление ответа

```haskell
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
```

Примеры задач на обработку списков

Примеры задач на обработку списков

Числа Фибоначчи

Числа Фибоначчи

Определение

Числами Фибоначчи называется числовая последовательность, задаваемая правилами:

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

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

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`

Примеры задач на обработку списков

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.

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.

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
```

Примеры задач на обработку списков

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

Примеры задач на обработку списков

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)
```

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"
```

Шифр Цезаря

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

```haskell
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

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

Список подходящих дробей

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

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: $e_n = 1 + r_n$.

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)
```

```
ghci> answer
153
```