Модуль 2 · Часть 1

Списки, кортежи и List Comprehensions

Курс «Функциональное и логическое программирование на Haskell»

Что мы изучим сегодня

Задание: генерация списков с помощью list comprehensions

1

Простые числа до N

Генерация последовательности простых чисел с помощью фильтрации

2

Пифагоровы тройки

Поиск всех (a, b, c) где $a^2 + b^2 = c^2$ через генераторы и условия

3

Матрица смежности

Построение графовых структур с помощью list comprehensions

List comprehensions — мощный инструмент для декларативной генерации данных

Что такое List Comprehension?

Математическая нотация для определения множеств

Математическая запись

$S = \{ x^2 \mid x \in \{0..5\}, x \text{ even} \}$

S = {0, 4, 16}

Haskell-запись

```
[x^2 | x <- [0..5], even x]
-- [0, 4, 16]
```

Синтаксис list comprehension

```
[ выражение | генератор, условие ]
```

- **выражение** — что генерируем (результат)
- **генератор** — откуда берём значения (x <- [1..n])
- **условие** — фильтр (guard), необязательно

Задача 1: Простые числа до N

Подход

Число p простое, если оно делится только на 1 и на себя.

Проверка: ни одно число от 2 до p-1 не делит p нацело.

Декомпозиция

Генератор: `p <- [2..n]`
Условие: нет делителей p
Результат: само число p

Реализация на Haskell

```
-- Простые числа до N
primesUpTo :: ???


-- > primesUpTo 30
-- [2,3,5,7,11,13,17,19,23,29]
```

Задача 1: Простые числа до N

Подход

Число p простое, если оно делится только на 1 и на себя.

Проверка: ни одно число от 2 до p-1 не делит p нацело.

Декомпозиция

Генератор: `p <- [2..n]`
Условие: нет делителей p
Результат: само число p

Реализация на Haskell

```haskell
-- Простые числа до N
primesUpTo :: Int -> [Int]
primesUpTo n = [p | p <- [2..n], null [d | d <- [2..p-1], p `mod` d == 0]]

-- > primesUpTo 30
-- [2,3,5,7,11,13,17,19,23,29]
```

Как это работает?

Пошаговый разбор генерации простых чисел

1

`Генератор p <- [2..n]`

Перебираем все числа от 2 до n. Каждое — кандидат в простые.

2

`Внутренний список делителей`

[d | d <- [2..p-1], p `mod` d == 0]
Находим все делители числа p.

3

`Фильтр: null (пустой список)`

Если список делителей пуст — число простое. null [] == True

4

`Результат: p`

Число p попадает в итоговый список простых чисел.

Оптимизация: достаточно проверять делители до √p — это уменьшит сложность

Задача 2: Пифагоровы тройки

Найти все (a, b, c), где $a^2 + b^2 = c^2$ и $a \le b \le c \le n$

Несколько генераторов

List comprehension может содержать несколько генераторов — они работают как вложенные циклы.

a ≤ b ≤ c исключает дубликаты.

Структура

Генераторы: `c, b, a <- [1..n]`
Условие: `a² + b² == c²`
Результат: кортеж `(a, b, c)`

Реализация на Haskell

```
-- Пифагоровы тройки до n
pyTriples :: ???
```

Задача 2: Пифагоровы тройки

Найти все (a, b, c), где $a^2 + b^2 = c^2$ и $a \le b \le c \le n$

Несколько генераторов

List comprehension может содержать несколько генераторов — они работают как вложенные циклы.

a ≤ b ≤ c исключает дубликаты.

Структура

Генераторы: `c, b, a <- [1..n]`
Условие: `a² + b² == c²`
Результат: кортеж `(a, b, c)`

Реализация на Haskell

```haskell
-- Пифагоровы тройки до n
pyTriples :: Int -> [(Int, Int, Int)]
pyTriples n =
  [(a, b, c) | c <- [1..n],
               b <- [1..c],
               a <- [1..b],
               a^2 + b^2 == c^2]
```

Трассировка: pyTriples 15

Как Haskell перебирает комбинации

c	b	a	a² + b²	c²	Совпадение?
5	4	3	25	25	✓
10	8	6	100	100	✓
13	12	5	169	169	✓
15	12	9	225	225	✓

```
> pyTriples 15

[(3,4,5),(6,8,10),(5,12,13),(9,12,15)]
```

Порядок генераторов важен: c — внешний цикл, a — внутренний. Это определяет порядок обхода.

Задача 3: Матрица смежности

Построение представления графа через list comprehensions

Что такое матрица смежности?

Двумерная матрица $n \times n$, где элемент $(i, j) = 1$, если существует ребро между вершинами i и j, иначе 0.

List comprehension позволяет построить её декларативно.

Пример: граф с 4 вершинами

	1	2	3	4
1	0	**1**	**1**	0
2	**1**	0	0	**1**
3	**1**	0	0	**1**
4	0	**1**	**1**	0

Реализация на Haskell

```
-- Матрица смежности из списка рёбер
type Edge = (Int, Int)
adjMatrix :: ???
```

Задача 3: Матрица смежности

Построение представления графа через list comprehensions

Что такое матрица смежности?

Двумерная матрица $n \times n$, где элемент $(i, j) = 1$, если существует ребро между вершинами i и j, иначе 0.

List comprehension позволяет построить её декларативно.

Пример: граф с 4 вершинами

	1	2	3	4
1	0	**1**	**1**	0
2	**1**	0	0	**1**
3	**1**	0	0	**1**
4	0	**1**	**1**	0

Реализация на Haskell

```haskell
-- Матрица смежности из списка рёбер
type Edge = (Int, Int)
adjMatrix :: Int -> [Edge] -> [[Int]]
adjMatrix n edges =
  [[if (i,j) `elem` edges || (j,i) `elem` edges
      then 1 else 0
   | j <- [1..n]] | i <- [1..n]]
```

Полезные паттерны

Часто встречающиеся приёмы с list comprehensions

Декартово произведение

```
pairs n = [(x,y) | x <- [1..n],
                   y <- [1..n]]
```

Фильтрация с условием

```
evens xs = [x | x <- xs,
                even x]
```

Преобразование элементов

```
toUpper' s = [toUpper c |
              c <- s]
```

Вложенные списки (flatten)

```
flatten xss = [x | xs <- xss,
                   x <- xs]
```

Бонус: Оптимизация простых чисел

Наивный подход: O(n²)

```
primesUpTo n =
  [p | p <- [2..n],
   null [d | d <- [2..p-1],
         p `mod` d == 0]]
```

Оптимизация: ?

```
primesOpt n =
```

Бонус: Оптимизация простых чисел

Наивный подход: O(n²)

```
primesUpTo n =
  [p | p <- [2..n],
   null [d | d <- [2..p-1],
         p `mod` d == 0]]
```

Оптимизация: O(n√n)

```
primesOpt n =
  [p | p <- [2..n],
   null [d | d <- [2..sqrtP p],
         p `mod` d == 0]]
  where sqrtP = floor . sqrt .
                fromIntegral
```

Почему достаточно проверять до √p?

Если p = a × b и a ≤ b, то a ≤ √p. Значит, если у p есть делитель больше √p, то есть и делитель меньше √p. Поэтому достаточно проверить только малые делители.

Итоги занятия

- List comprehension — мощный инструмент для генерации списков декларативным способом
- Синтаксис: [выражение | генератор, условие] напоминает математическую нотацию
- Несколько генераторов работают как вложенные циклы
- Простые числа, пифагоровы тройки и матрицы — отличная практика
- Оптимизация: проверка делителей до √n вместо n

Следующее занятие: **Рекурсия — хвостовая и обычная, классические задачи**

Вопросы?

Модуль 2, Часть 1: List Comprehensions