МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление 02.03.01 Математика и компьютерные науки

Отчет по лабораторной работе №2

по дисциплине «Функциональное программирование»

Вариант 19

Обучающийся: _____ Шихалев А.О.

Руководитель: _____ Моторин Д.Е.

«_____»_____20____г.

Санкт-Петербург, 2024

Содержание

Введение

Практическое задание №2 представляет собой реализацию двух программ:

1. Вычислить все пары координат (x, y) для заданного фрактала на глубину n шагов и вывести в виде списка списков пар (каждый уровень рекурсии является списком пар).

Фрактал: Папоротник Барнсли

2. Реализовать заданную игру и стратегию следующим образом: в коде задать список, содержащий ходы пользователя; реализовать функцию, определяющую работу стратегии, и функцию, организующую игру (игра должна продолжаться не более 100 ходов).

Игра: Ним (N кучек в каждой не более M камней, игрок может взять неограниченное число камней)

Лабораторная работа выполнена на языке Haskell в текстовом редакторе Visual Studio Code 1.95.3.

1 Математическое описание

1.1 Фракталы

Фрактал — множество, обладающее свойством самоподобия (объект, в точности или приближённо совпадающий с частью себя самого, то есть целое имеет ту же форму, что и одна или более частей). В математике под фракталами понимают множества точек в евклидовом пространстве, имеющие дробную метрическую размерность, либо метрическую размерность, отличную от топологической, поэтому их следует отличать от прочих геометрических фигур, ограниченных конечным числом звеньев. Самоподобные фигуры, повторяющиеся конечное число раз, называются предфракталами.

1.2 Папоротник Барнсли

Папоротник Барнсли — фрактал, названый в честь британского математика Майкла Барнсли, впервые описан в его книге «Fractals Everywhere» [**?**]. Для его построения используется четыре простых трансформации, которые применяются случайным образом с определёнными вероятностями. Каждое преобразование масштабирует, поворачивает и смещает точки, начиная с одного начального положения. На каждом шаге, в зависимости от случайно выбранной трансформации, новая точка генерируется на основе предыдущей, и этот процесс повторяется многократно. Результатом становится изображение, напоминающее настоящие папоротники.

$$
\begin{aligned}
T_1(x,y) &= (0, 0.16y)\,, \\
T_2(x,y) &= (0.85x + 0.04y, -0.04x + 0.85y + 1.6)\,, \\
T_3(x,y) &= (0.2x - 0.26y, 0.23x + 0.22y + 1.6)\,, \\
T_4(x,y) &= (-0.15x + 0.28y, 0.26x + 0.24y + 0.44)\,.
\end{aligned}
$$

Каждое из этих преобразований применяется с вероятностью:

$$P(T_1) = 0.01, \quad P(T_2) = 0.85, \quad P(T_3) = 0.07, \quad P(T_4) = 0.07.$$

На рисунках 1-3 приведены примеры папоротника Барнсли для разного количества точек (n). Во всех примерах в качестве начальной была выбрана точка с координатами (0, 0).

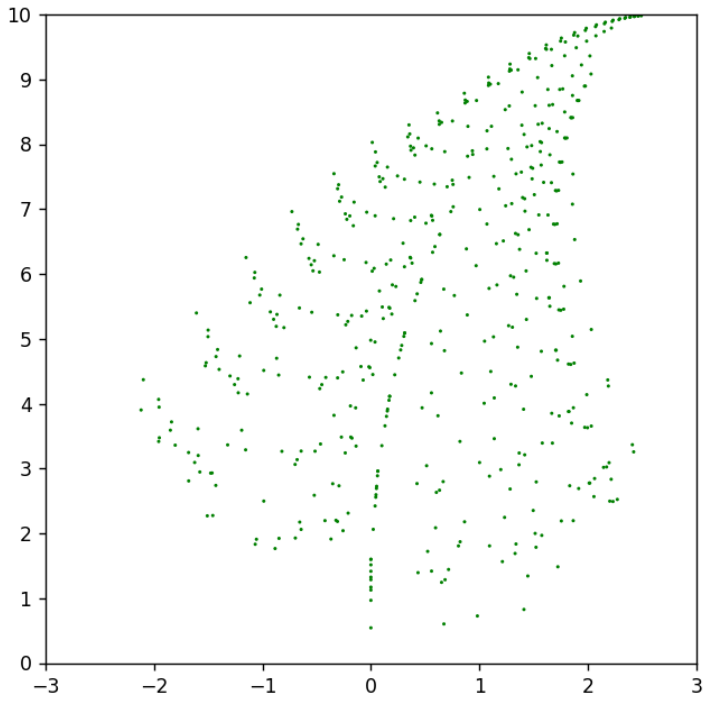

Рис. 1. Папоротник Барнсли для n = 500.

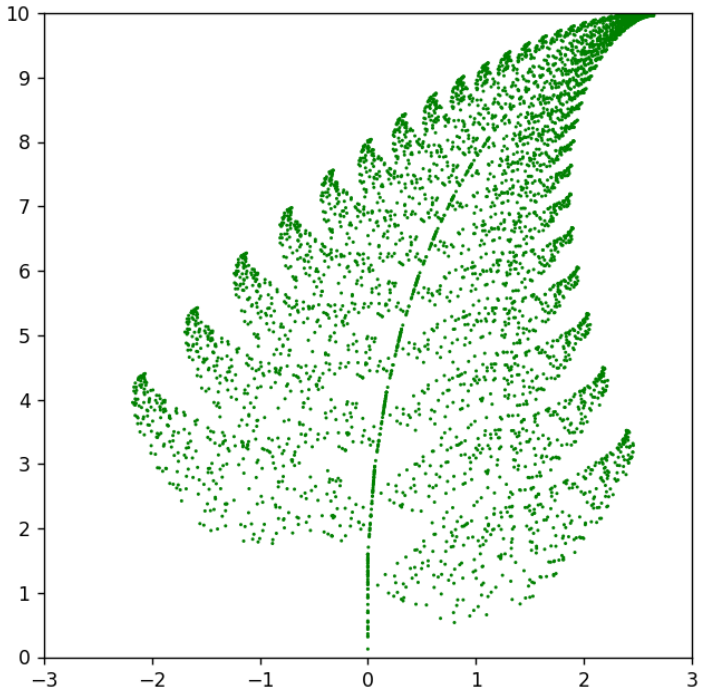

Рис. 2. Папоротник Барнсли для n = 5000.

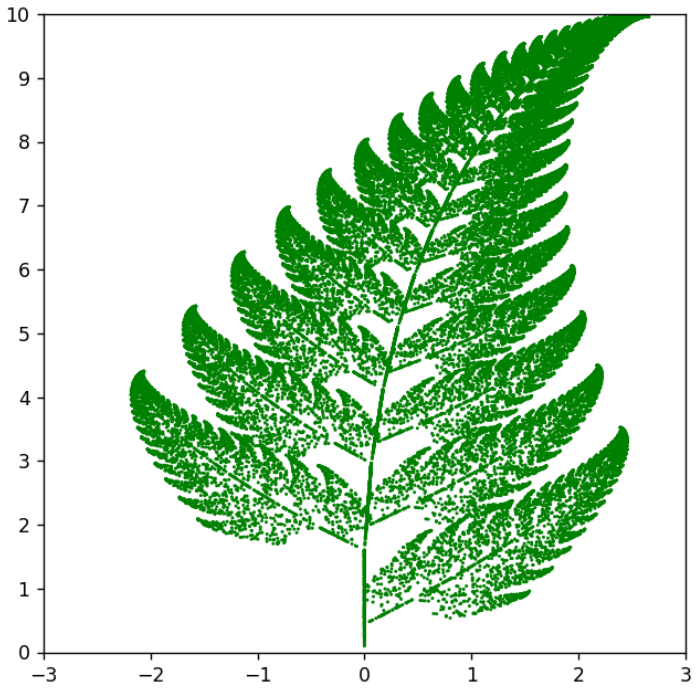

Рис. 3. Папоротник Барнсли для n = 50000.

1.3 Игра Ним

Ним - игра, в которой два игрока по очереди берут предметы, разложеннные на несколько кучек. За один ход может быть взято любое количество предметов (больше нуля) из одной кучки. Выигрывает игрок, взявший последний предмет. В классическом варианте игры число кучек равняется трем. В нашем варианте кучек может быть произвольное число, не больше числа N. В общем случае рассматривается p кучек камней с $N_1, N_2, ...N_p$ камнями. Игроки ходят по очереди. Ход заключается в том, что игрок берёт из кучки i $\in$ [1,p], n $\in [1, N_i]$ предметов. Каждой позиции игры ставится в соответствие **ним-сумма** этой позиции — результат сложения размеров всех кучек в двоичной системе счисления без учёта переноса разрядов, то есть сложение двоичных разрядов чисел в поле вычетов по модулю 2: S = $N_1 \oplus N_2 \oplus ... \oplus N_p$

Выигрышная стратегия состоит в том, чтобы оставлять после своего хода позицию с ним-суммой, равной нулю. Она основана на том, что из любой позиции с ним-суммой, не равной нулю, можно одним ходом получить позицию с нулевой ним-суммой, а из позиции с нулевой ним-суммой любой ход ведёт в позицию с ним-суммой, отличной от нуля.

2 Особенности реализации

2.1 Папоротник Барнсли

2.1.1 Аффинные преобразование точек

Функции, код которых представлен в листинге 1, реализуют четыре аффинных преобразования, используемых для построения фрактала папоротника Барнсли. Каждое преобразование принимает на вход точку в двумерном пространстве (координаты x и y) и возвращает новую точку, которая является результатом применения соответствующего преобразования.

Листинг 1. Аффинные преобразование точек

```haskell
type Point = (Double, Double)

transformation1 :: Point -> Point
transformation1 (_, y) = (0, 0.16 * y)

transformation2 :: Point -> Point
transformation2 (x, y) = (0.85 * x + 0.04 * y, -0.04 * x + 0.85 * y + 1.6)

transformation3 :: Point -> Point
transformation3 (x, y) = (0.2 * x - 0.26 * y, 0.23 * x + 0.22 * y + 1.6)

transformation4 :: Point -> Point
transformation4 (x, y) = (-0.15 * x + 0.28 * y, 0.26 * x + 0.24 * y + 0.44)
```

2.1.2 Генерация новой точки

Генерация новой точки происходит с помощью функции `generateNextPoint` и вспомогательной функции `chooseTransformation`, код которых представлен в листинге 2. `chooseTransformation` принимает на вход исходную точку и случайное число от 0 до 1, затем выбирает и применяет к точке трансформацию в соответствии с заданными вероятностями, и возвращает новую точку. `generateNextPoint` принимает на вход исходную точку и состояние генератора случайных чисел – `g`, генерирует случайное число от 0 до 1, применяет функцию `applyTransformation` и возвращает пару – новую точку и новое состояние генератора случайных чисел.

Листинг 2. Код функций для генераций новых точек в папоротнике Барнсли.

```haskell
import System.Random (randomR, mkStdGen, RandomGen, StdGen)

chooseTransformation :: Point -> Double -> Point
chooseTransformation point randValue
| randValue < 0.01 = transformation1 point
| randValue < 0.86 = transformation2 point
| randValue < 0.93 = transformation3 point
| otherwise = transformation4 point

generateNextPoint :: RandomGen g => Point -> g -> (Point, g)
generateNextPoint point gen =
let (randValue, newGen) = randomR (0.0, 1.0) gen  -- генерируем случайное ч
    ↪исло
in (chooseTransformation point randValue, newGen)
```

2.1.3 Рекурсивная генерация папоротника Барнсли

Функция `barnsleyFern`, код которой представлен в листинге 3, реализует рекурсивный алгоритм генерации списка точек, из которых состоит папоротник Барнсли. Функция принимает на вход текущее состояние генератора случайных чисел, начальную точку и число – количество шагов рекурсии, а возвращает список точек папоротника Барнсли.

Листинг 3. Код функции для построения папоротника Барнсли.

```haskell
buildFern :: RandomGen g => Point -> Int -> g -> [Point]
buildFern _ 0 _ = []
buildFern currentPoint steps gen =
let (nextPoint, newGen) = generateNextPoint currentPoint gen
remainingPoints = buildFern nextPoint (steps - 1) newGen
in currentPoint : remainingPoints
```

2.2 Реализация игры «Ним»

2.2.1 Инициализация игры

Функция `initializeGame`, представлена в листинге листинге 4, позволяет задать начальное состояние игры. Пользователь вводит количество куч и количество камней в каждой из них. Если ввод данных не совпадает с указанным количеством куч, функция вызывает себя рекурсивно, чтобы исправить ошибку ввода.

Листинг 4. Инициализация игры

```haskell
initializeGame :: IO GameState
initializeGame = do
```

```
putStrLn "Введите количество куч:"
nPiles <- read <$> getLine
putStrLn "Введите количество камней в каждой куче (через пробел):"
stones <- map read . words <$> getLine
if length stones == nPiles
then return stones
else do
putStrLn "Ошибка: количество куч не совпадает с введенными значениями."
initializeGame
```

2.2.2 Проверка окончания игры

Функция `isGameOver`, приведенная в листинге листинге 5, проверяет, завершена ли игра, возвращая `True`, если все кучи пусты.

Листинг 5. Проверка окончания игры

```
isGameOver :: GameState -> Bool
isGameOver = all (== 0)
```

2.3 Ходы игроков

Функция `playerMove`, представлена в листинге листинге 6, позволяет пользователю сделать ход. Игрок выбирает кучу и количество камней, которые хочет взять. Если введенные данные некорректны, функция повторяет запрос ввода.

Листинг 6. Ход игрока

```
playerMove :: GameState -> IO (Int, Int)
playerMove piles = do
printGameState piles
putStrLn "Выберите номер кучи (начиная с 1):"
pile <- read <$> getLine
putStrLn "Сколько камней вы хотите взять?"
stones <- read <$> getLine
if pile > 0 && pile <= length piles && stones > 0 && stones <= piles !! (
    ↪pile - 1)
then return (pile - 1, stones)
else do
putStrLn "Неверный ход. Попробуйте снова."
playerMove piles
```

Функция `computerMove`, представлена в листинге 7, реализует стратегию компьютера на основе Nim-суммы. Если Nim-сумма равна нулю, компьютер выбирает любой допусти-

мый ход. В противном случае он совершает выигрышный ход, уменьшая количество камней в одной из куч.

Листинг 7. Ход компьютера

```
computerMove :: GameState -> IO (Int, Int)
computerMove piles = do
let nimSum = foldr xor 0 piles
let moves = [(i, 1) | (i, s) <- zip [0..] piles, s > 0]
let winningMoves = [(i, s - (s `xor` nimSum)) | (i, s) <- zip [0..] piles,
    ↪s `xor` nimSum < s]

let move = if nimSum == 0
then case moves of
(x:_) -> x
[]    -> error "Нет возможных ходов"
else case winningMoves of
(x:_) -> x
[]    -> error "Нет выигрышных ходов"

let (pile, stones) = move
putStrLn $ "Компьютер выбрал кучу " ++ show (pile + 1) ++ " и взял " ++
    ↪show stones ++ " камней."
return move
```

2.3.1 Обновление состояния игры

Функция `makeMove`, приведенная в листинге листинге 8, обновляет состояние игры после хода, уменьшая количество камней в выбранной куче.

Листинг 8. Обновление состояния игры

```
makeMove :: GameState -> (Int, Int) -> GameState
makeMove piles (pile, stones) =
take pile piles ++ [piles !! pile - stones] ++ drop (pile + 1) piles
```

2.4 Основная логика игры

Функция `playNim`, приведенная в листинге 9, организует игровой процесс. Она последовательно обрабатывает ходы игрока и компьютера, обновляет состояние игры и завершает игру, когда все кучи пусты.

Листинг 9. Основная логика игры

```
playNim :: GameState -> MoveHistory -> IO ()
playNim piles moves
```

```
3  | isGameOver piles = do
4  putStrLn "Игра окончена!"
5  printMoveHistory moves
6  | otherwise = do
7  (pile, stones) <- playerMove piles
8  let newPiles = makeMove piles (pile, stones)
9  let newMoves = moves ++ [("Вы", pile, stones)]
10 if isGameOver newPiles
11 then do
12 putStrLn "Поздравляем, вы выиграли!"
13 printMoveHistory newMoves
14 else do
15 (pileComp, stonesComp) <- computerMove newPiles
16 let finalPiles = makeMove newPiles (pileComp, stonesComp)
17 let finalMoves = newMoves ++ [("Компьютер", pileComp, stonesComp)]
18 if isGameOver finalPiles
19 then do
20 putStrLn "Компьютер выиграл!"
21 printMoveHistory finalMoves
22 else playNim finalPiles finalMoves
```

3 Результаты работы программы

На рисунке 4 демонстрируется работа программы для генерации точек папоротника Барнсли на примере с 20 точками.

На рисунке 5 представлена работа программы игры Ним.

```
Введите количество шагов для генерации фрактала:
20
[(0.0,0.0),(0.0,0.44),(1.76e-2,1.9740000000000002),(-0.5097200000000001,2.038328),(-0.3517
3.3529676000000004),(-0.1648508440000002,4.464091615200001),(3.844044720800002e-2,5.40107
8),(0.24871725639400002,6.18937350278968),(0.4589846080464872,6.851018787115468),(0.664177
41328,7.405006584726287),(0.860751281464564 4,7.867688490284378),(1.0463461288562548,8.2531
483138),(1.219518416147142,8.573285455064 17),(-1.9851505586022402,3.7666120557252545),(-1
13492582894,4.881026269710556),(-1.1109654179070376,5.810340868957288),(-0.711906970462690
83228355329977),(-0.34179179068008775,7.224220380848989),(-1.5542068441150025e-3,7.7542589
843),(0.308849283996456,8.191182314320281)]
ghci>
```

Рис. 4. Результат запуска программы для генерации точек папоротника Барнсли

```
ghci> main
Добро пожаловать в игру Ним!
Введите количество куч:
2
Введите количество камней в каждой куче (через пробел):
7 8
Кучки: 1: 7, 2: 8
Выберите номер кучи (начиная с 1):
1
Сколько камней вы хотите взять?
3
Компьютер выбрал кучу 2 и взял 4 камней.
Кучки: 1: 4, 2: 4
Выберите номер кучи (начиная с 1):
1
Сколько камней вы хотите взять?
3
Компьютер выбрал кучу 2 и взял 3 камней.
Кучки: 1: 1, 2: 1
Выберите номер кучи (начиная с 1):
1
Сколько камней вы хотите взять?
1
Компьютер выбрал кучу 2 и взял 1 камней.
Компьютер выиграл!!!!!!
Ходы в игре:
Вы взяли 3 камней из кучи 1
Компьютер взяли 4 камней из кучи 2
Вы взяли 3 камней из кучи 1
Компьютер взяли 3 камней из кучи 2
Вы взяли 1 камней из кучи 1
Компьютер взяли 1 камней из кучи 2
ghci>
```

Рис. 5. Результат работы программы игры Ним

Заключение

В ходе выполнения лабораторной работы были реализованы две программы на языке программирования Haskell. Первая программа генерирует точки для папоротника Барнсли с помощью рекурсивного алгоритма с заданным количеством шагов рекурсии. Вторая программа реализует классическую игру «Ним», в которой игрок и компьютер поочередно делают ходы. Для каждой задачи был сформирован (.hs) файл, содержащий реализацию.

Список литературы

[1] Курт У. Программируй на Haskell / пер. с англ. С. Соловьева. — Москва: ДМК Пресс, 2019. — 384 с.

[2] Barnsley M.F. Fractals Everywhere. — Academic Press, 1988. — 394 p. ISBN-13 978-0-12-079061-6

[3] Bouton C.L. Nim, a game with a complete mathematical theory // Annals of Mathematics. — 1901. — Vol. 3, No. 1. — P. 35–39. DOI 10.2307/1967631.