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 практической работе №2

по дисциплине «Функциональное программирование»

Обучающийся: _____ Гладков И.А.

Руководитель: _____ Моторин Д.Е.

«_____»_____20____г.

Санкт-Петербург, 2024

Содержание

Введение

В данном отчете, описаны результаты выполнения практических заданий: реализации фрактала дерево Пифагора и игры Ним. В ходе выполнения практической работы, реализованы на языке Haskell. Постановка задач:

1. Фрактал:

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

2. Игра:

   Реализовать заданную игру и стратегию следующим образом: в коде задать список, содержащий историю ходов; реализовать функцию, определяющую работу стратегий и функцию, организующую игру. Вариант игры: Ним (одна кучка из M камней, игрок может взять не более K камней).

Для каждой части задачи необходимо сформировать (.hs) файл, содержащий весь необходимый код на языке Haskell.

1 Задание 1: дерево Пифагора

1.1 Теоретические сведения

Фрактал – множество, обладающее свойством самоподобия (объект, в точности или приближенно совпадающий с частью себя самого, то есть целое имеет ту же форму, что и одна или более частей).

Дерево Пифагора — разновидность фрактала, основанная на фигуре, известной как «Пифагоровы штаны».

Обнаженное дерево Пифагора — разновидность фрактала, которая получается, если изображать только отрезки, соединяющие «центры» треугольников «Пифагоровых штанов».

Обнажённое дерево Пифагора строится рекурсивно следующим образом:

1. Строится вертикальный отрезок.
2. Из верхнего конца этого отрезка рекурсивно строятся ещё два отрезка меньшей длины (длина предыдущей ветки умноженная на $\sqrt{2}/2$)под углом 90° друг к другу, и по 135° по отношению к предыдущей ветке (в сумме 360°).
3. Для каждой ветви дерева вызывается функция построения двух последующих отрезков.

1.2 Реализация

Ниже приведен код, на языке Haskell, выводящий список списков пар координат, в зависимости от введенного пользователя значения (глубины рекурсии).

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

nextLevel :: Line -> [Line]
nextLevel ((x1,y1), (x2, y2)) =
  let dx = x2 - x1
    dy = y2 - y1
    len = (sqrt (dx^2 + dy^2) / sqrt 2)
    deg1 = atan2 dy dx + pi / 4
    deg2 = atan2 dy dx - pi / 4
    p1 = (x2 + len * cos deg1, y2 + len * sin deg1)
    p2 = (x2 + len * cos deg2, y2 + len * sin deg2)
  in [((x2,y2),p1), ((x2,y2), p2)]

generateFractal :: Int -> [Line]
generateFractal 0 = [((2,0), (2,1))]
generateFractal n =
  let prevLevel = generateFractal (n-1)
  in concatMap nextLevel prevLevel

groupByLevels :: Int -> [[Line]]
```

```
groupByLevels 0 = [[((2,0),(2,1))]]
groupByLevels n =
  let prevLevels = groupByLevels (n-1)
    nextLines = concatMap nextLevel (last prevLevels)
  in  prevLevels ++ [nextLines]

printLn :: Show a => [[a]]->IO()
printLn = mapM_(putStrLn.show)

main :: IO()
main = do
  putStrLn "Введите глубину фрактала:"
  input <- getLine
  let steps = read input :: Int
  printLn (groupByLevels steps)
```

1.3 Результаты

В результате работы программы пользователю будут выведен список списков пар координат фрактала.

При вводе глубины фрактала 3 получаем следующие точки:

```
[((2.0,0.0),(2.0,1.0))]
[((2.0,1.0),(1.5,1.5)),((2.0,1.0),(2.5,1.5))]
[((1.5,1.5),(1.0,1.5)),((1.5,1.5),(1.5,2.0)),((2.5,1.5),(2.5,2.0)),((2.5,1.5),(3.0,1.5))]
[((1.0,1.5),(0.75,1.25)),((1.0,1.5),(0.75,1.75)),((1.5,2.0),(1.25,2.25)),((1.5,2.0)
    ↪,(1.75,2.25)),((2.5,2.0),(2.25,2.25)),((2.5,2.0),(2.75,2.25)),((3.0,1.5)
    ↪,(3.25,1.75)),((3.0,1.5),(3.25,1.25))]
```

При визуализации данного фрактала с глубиной рекурсии 3 и 10, получаем следующие изображения:

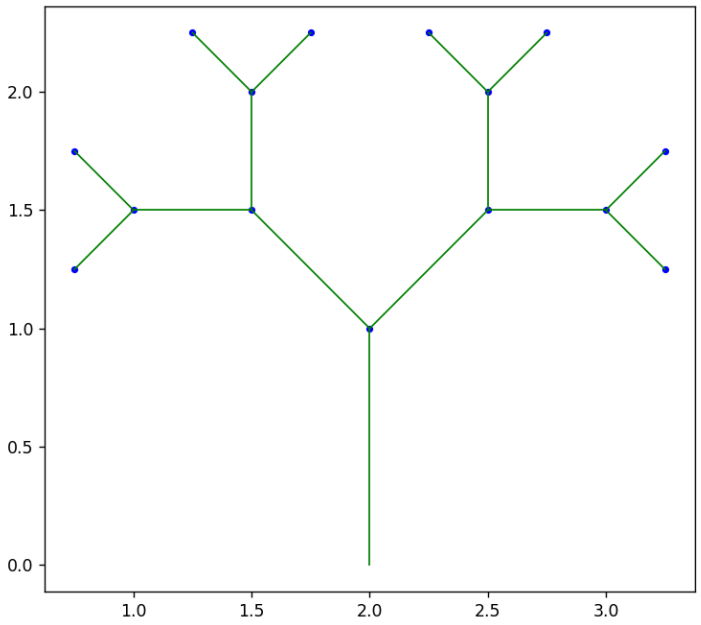

Рис. 1. Фрактал с глубиной рекурсии 3

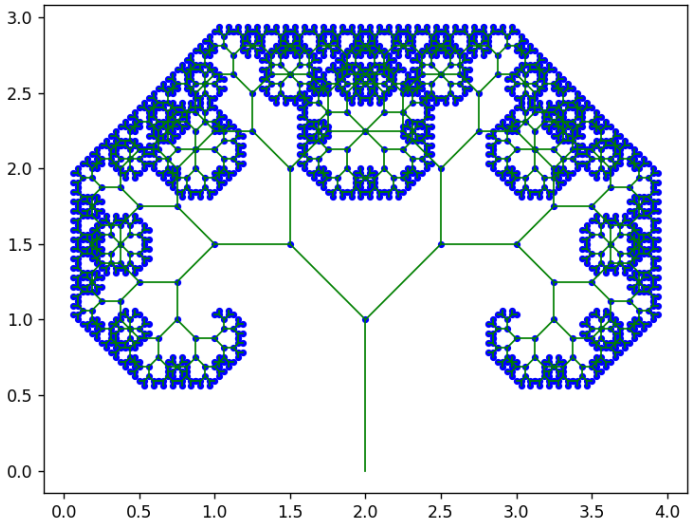

Рис. 2. Фрактал с глубиной рекурсии 10

Как можно заметить, точки, которые вывела программа, совпадают с точками на рисунке 1. Это говорит о корректности программы.

2 Задание 2: игра НИМ

2.1 Теоретические сведения

Ним - игра, в которой два игрока по очереди берут предметы, разложенные 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 кучка всего одна.

Выигрышной стратегией при таком варианте будут ходы, которые приводят кучу в состояние, когда оставшееся количество камней делится без остатка на (k+1), так как при таком состоянии мы сможем оставлять противнику позиции, в которых ему придется оставлять нам по итогу как минимум один камень. Если же к нашему ходу количество камней уже делится без остатка на (k+1), то мы находимся в проигрышной позиции и не сможем изменить ход игры, если противник не ошибется.

2.2 Реализация

Ниже приведен код, на языке Haskell, реализующий игру НИМ с параметрами которые необходимо ввести пользователю при запуске программы (N – количество камней в куче и K – ограничение на взятие камней за ход). Во время работы программы каждый ход пользователю пишется, сколько взял бот камней, и спрашивают сколько он собирается взять.

```haskell
{-# LANGUAGE FlexibleContexts #-}

type Step = (String,Int)
type Steps = [Step]


botStep :: (Int, Int, Int) -> (Int, Int, Int) -- (n, k, count)
botStep (n, k, count)
  | n < k = (n, k , n)
  | n > k && count == 1 = (n, k, 1)
  | otherwise =
    if (n - count) `mod` (k + 1) == 0 then (n, k, count)
    else botStep(n, k, count - 1)


check :: Int -> Int -> IO Int --[k, count]
check k count  =
  if count > 0 && count <= k then do return count
  else do
    putStrLn ("Ошибка ввода. Попробуйте снова")
    input <- getLine
    let pl = read input :: Int
    check k pl
```

```haskell

takeFirstOf3 :: (Int, Int, Int) -> Int
takeFirstOf3 (_,_,count) = count

takeFirstOf2 :: (String, Int) -> String
takeFirstOf2 (name,_) = name

step :: (Int, Int, Steps) -> IO (Int, Int, Steps)
step (n, k, history)
  | n == 0  = do
    if takeFirstOf2(last history) == "Bot" then
      putStrLn ("Бот выйграл")
    else
      putStrLn ("Вы выйграли!!")
    return (n, k, history)
  | n > 0 = do
    let bot = takeFirstOf3(botStep (n, k, k))
      botName = "Bot"
      playerName = "Player"
      newn = n - bot
      newhistory = history ++ [(botName, bot)]
    putStrLn ("Было "++ show n ++". Бот взял " ++ show bot ++ " камня(ей)")
    if newn == 0 then do
      putStrLn ("Бот выйграл")
      return (newn, k, newhistory)
    else do
      putStrLn ("Сколько камней возьмете вы? В куче сейчас " ++ show newn ++ ", ограничен
    ↪ие - " ++ show k)
      input <- getLine
      let pl = read input :: Int
      finalpl <- check k pl
      let newhistory2 = newhistory ++ [(playerName, finalpl)]
      step (newn - finalpl, k, newhistory2)

printLn :: [Step] -> IO()
printLn = mapM_ (\(name, count) -> putStrLn(name ++ ": " ++ show count))

main :: IO()
main = do
  putStrLn "Введите количество камней в куче:"
  input <- getLine
  let n = read input :: Int
  putStrLn "Введите ограничение на взятие камней:"
  input2 <- getLine
  let k = read input2 ::Int
  let history =[] :: [Step]
  (n', k',history') <- step (n, k, history)
```

```
  putStrLn ""
  putStrLn "История ходов:"
  printLn history'

```

2.3 Результаты

В качестве примера, возьмем количество камней в куче равным 15 (N = 15), ограничение на взятие камней равным 5 (K = 5). Затем на первом нашем ходу возьмем 5 камней, на следующем – 4, и так далее.

В конце выведется список ходов в игре.

```
Введите количество камней в куче:
15
Введите ограничение на взятие камней:
5
Было 15. Бот взял 3 камня(ей)
Сколько камней возьмете вы? В куче сейчас 12, ограничение - 5
5
Было 7. Бот взял 1 камня(ей)
Сколько камней возьмете вы? В куче сейчас 6, ограничение - 5
4
Было 2. Бот взял 2 камня(ей)
Бот выйграл

История ходов:
Bot: 3
Player: 5
Bot: 1
Player: 4
Bot: 2
```

Заключение

В ходе выполнения практической работы были написаны две программы, одна реализует дерево Пифагора, другая – игру НИМ. Для каждой задачи был сформирован (.hs) файл, содержащий реализацию.

Прогргамма реализующая дерево Пифагора выводит пары координат фрактала на заданную пользователем глубину

Программа реализующая игру НИМ выводит историю ходов. В ней реализована функции реализующие работу стратегии и организующие логику игры.

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.