МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего 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»

Введение в Haskell

Практическое задание 2

Студент,
группы 5130201/20101 _____ Астафьев И. Е.

Преподаватель _____ Моторин Д. Е.

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

Введение

В данном отчете, описаны результаты выполнения практических заданий: реализации фрактала кривой Пеано и реализации игры повторяющаяся дилемма заключенного. Программы реализуются 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: Кривая Пеано.

2. **Игра:**

   Реализовать заданную игру и стратегию следующим образом: В коде задать список, содержащий ходы пользователя; Реализовать функцию, определяющую работу стратегий и функцию, организующую игру (игра должна продолжатся не более 100 ходов). Вариант игры: Повторяющаяся дилемма заключенного (стратегия равновесие по Нэшу и «добрая» стратегия).

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

1 Задание 1: Кривая Пеано

1.1 Теоретические сведения

- Фрактал представляет собой геометрическую фигуру, обладающую самоподобием, то есть её структура воспроизводится на различных масштабах. Характерной чертой фракталов является их сложная форма, при этом они легко создаются с помощью рекурсивных процедур. Для описания фракталов используются рекурсивные алгоритмы, которые с каждым шагом добавляют новые детали. Ключевыми свойствами фракталов являются самоподобие, размерность и рекурсивная структура.

- В геометрии кривая Пеано является первой открытой заполняющей пространство кривой. Кривая Пеано представляет собой сюръективную, непрерывную функцию из единичного отрезка в единичный квадрат, однако она не является инъективной. Пеано вдохновился более ранним результатом Георга Кантора, который показал, что эти два множества имеют одинаковую мощность. Некоторые авторы используют выражение «кривая Пеано» более широко, чтобы обозначать любую заполняющую пространство кривую.

  Кривая Пеано строится рекурсивно, путем деления квадрата на подквадраты, и каждый подквадрат заполняется модифицированной версией кривой предыдущего уровня. Построение кривой Пеано для уровня $n \geq 1$ осуществляется следующим образом. Кривая P_n строится из P_{n-1}. Квадрат размером $(3^n \times 3^n)$ делится на 9 подквадратов размером $(3^{n-1} \times 3^{n-1})$. Каждое из этих подразделений заполняется копией кривой, с определенными модификациями:

  1. Верхний левый квадрат: P_{n-1}.
  2. Верхний средний квадрат: P_{n-1} поворачивается на $90°$ против часовой стрелки и перемещается вправо на 3^{n-1}.
  3. Верхний правый квадрат: P_{n-1} перемещается вправо на $2 \times 3^{n-1}$.
  4. Средний правый квадрат: P_{n-1} перемещается вправо на $2 \times 3^{n-1}$ и вниз на 3^{n-1}.
  5. Средний квадрат: P_{n-1} поворачивается на $90°$ по часовой стрелке и перемещается вправо на 3^{n-1} и вниз на 3^{n-1}.
  6. Средний левый квадрат: P_{n-1} перемещается вниз на 3^{n-1}.
  7. Нижний левый квадрат: P_{n-1} перемещается вниз на $2 \times 3^{n-1}$.
  8. Нижний средний квадрат: P_{n-1} поворачивается на $90°$ против часовой стрелки и перемещается вправо на 3^{n-1} и вниз на $2 \times 3^{n-1}$.
  9. Нижний правый квадрат: P_{n-1} перемещается вправо на $2 \times 3^{n-1}$ и вниз на $2 \times 3^{n-1}$.

- L-система или система Линденмайера — это параллельная система переписывания и вид формальной грамматики. L-система состоит из алфавита символов, которые могут быть использованы для создания строк, набора порождающих правил, которые задают правила подстановки вместо каждого символа, начальной строки («аксиомы»), с которой начинается построение, и механизма перевода образованной строки в геометрические структуры.

  В описанной L-системе используются:

- Переменные: L, R, F
- Константы: $+$, $-$
- Начальная строка: L
- Правила переписывания:
  - $L \to LFRFL + F + RFLFR - F - LFRFL$
  - $R \to RFLFR - F - LFRFL + F + RFLFR$

Здесь:

- "F"указывает на движение вперед.
- "+"и "−"обозначают вращение на 90° по часовой и против часовой стрелки соответственно.

В этом подходе к генерации кривой используется текстовая строка, последовательно трансформируемая согласно правилам переписывания, из которых создается чертеж кривой. В каждом шаге осуществляются переходы и повороты, что в итоге приводит к построению кривой Пеано. Каждый новый уровень итерации добавляет больше деталей рисунку кривой.

1.2 Реализация

Ниже приведен код, на языке Haskell, выводящий списки списков пар координат в зависимости от введенного числа, которая в реализации означает глубину рекурсии.

```haskell
import System.IO

-- Функция для генерации строки команд после n итераций
derive :: Int -> [Char]
derive 0 = "L"
derive n = applyRules (derive (n - 1))

-- Применение правил к каждому символу в строке
applyRules :: [Char] -> [Char]
applyRules s = concatMap applyRule s

-- Правила L-системы для кривой Пеано
applyRule :: Char -> [Char]
applyRule 'L' = "LFRFL+F+RFLFR-F-LFRFL"
applyRule 'R' = "RFLFR-F-LFRFL+F+RFLFR"
applyRule c   = [c]  -- Оставляем другие символы без изменений

-- Интерпретация строки команд в список координат (x, y)
interpret :: Int -> [Char] -> [(Double, Double)]
interpret n commands = initialPos : eval commands initialPos initialAngle
  where
    initialPos = (0.5 * step, 0.5 * step)   -- Начальная позиция
    initialAngle = 0    -- Начальный угол (в градусах)
    step = 1 / (fromIntegral (3^n))   -- Шаг, зависящий от глубины рекурсии
    eval :: [Char] -> (Double, Double) -> Double -> [(Double, Double)]
```

```haskell
    eval [] _ _ = []
    eval (c:cs) (x,y) angle
      | c == 'F' = let dx = step * cos (angle * pi / 180)
                       dy = step * sin (angle * pi / 180)
                       x' = x + dx
                       y' = y + dy
                   in (x', y') : eval cs (x', y') angle
      | c == '+' = eval cs (x, y) (angle + 90)  -- Поворот влево на 90 градусов
      | c == '-' = eval cs (x, y) (angle - 90)  -- Поворот вправо на 90 градусов
      | otherwise = eval cs (x, y) angle    -- Игнорируем другие символы

-- Главная функция, вычисляющая координаты кривой Пеано
peanoCurve :: Int -> [(Double, Double)]
peanoCurve n = interpret n (derive n)

main :: IO ()
main = do
  let n = 4  -- Максимальная глубина рекурсии
  let initialPos = [(0.5, 0.5)]  -- Список с начальной позицией
  -- Генерируем список списков координат для каждой глубины от 1 до n
  let pointsList = initialPos : [ peanoCurve k | k <- [1..n] ]

  putStrLn $ "n=" ++ show n
  out <- openFile "C:/Users/iasta/peano_fractal.txt" WriteMode
  hPrint out pointsList
  hClose out
  -- print pointsList
```

1.3 Результаты

В результате выполнения программы будут выведены списки списков всех координат. Для наглядности отрисуем последний список списка для уровней n=2 и n=4 (Рис. 1, 2).

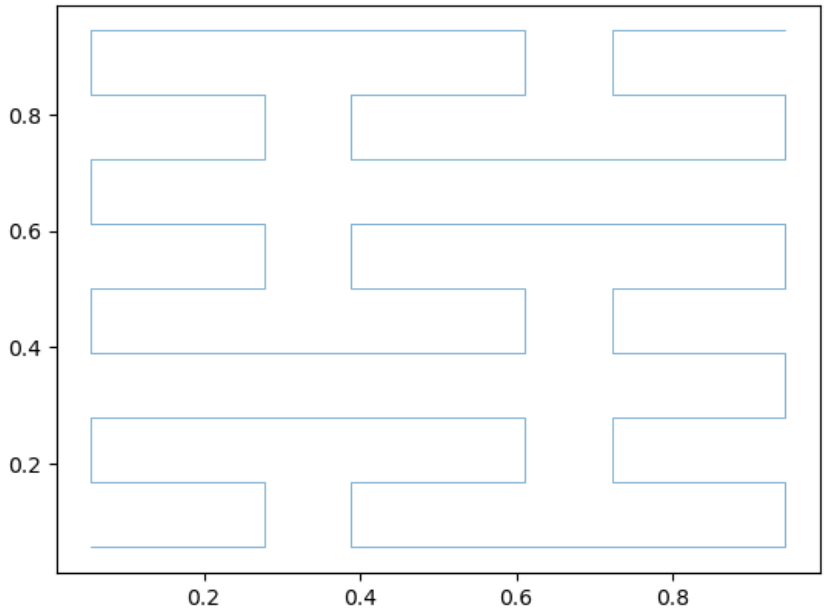

Рис. 1. Результат при n=2

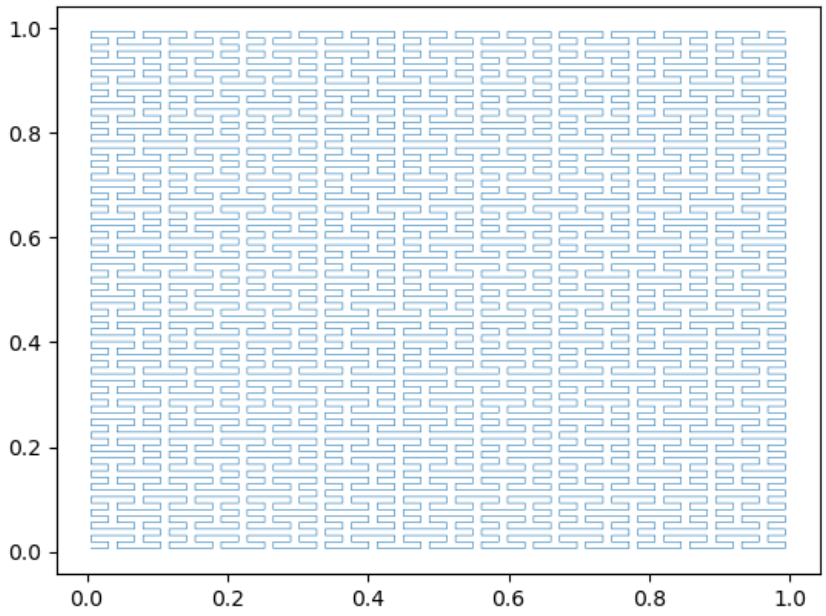

Рис. 2. Результат при n=5

2 Задание 2: Повторяющаяся дилемма заключенного

2.1 Теоретические сведения

- Дилемма заключённого - фундаментальная проблема в теории игр, согласно которой рациональные игроки не всегда будут сотрудничать друг с другом, даже если это в их интересах. Предполагается, что игрок («заключённый») максимизирует свой собственный выигрыш, не заботясь о выгоде других.
- В книге «Эволюция кооперации» 1984 года Роберт Аксельрод исследовал расширение сценария дилеммы, которое он назвал повторяющаяся дилемма заклю-

чённого (ПДЗ). В ней участники делают выбор снова раз за разом и помнят предыдущие результаты.

- Анализируя стратегии, набравшие лучшие результаты, Аксельрод назвал несколько условий, необходимых, чтобы стратегия получила высокий результат:

  – Добрая. Важнейшее условие — стратегия должна быть «доброй», то есть не предавать, пока этого не сделает оппонент. Почти все стратегии-лидеры были добрыми. Поэтому чисто эгоистичная стратегия по чисто эгоистическим причинам не будет первой «бить» соперника.

  – Мстительная. Успешная стратегия не должна быть слепым оптимистом. Она должна всегда мстить. Пример прощающей стратегии — всегда сотрудничать. Это очень плохой выбор, поскольку «подлые» стратегии воспользуются этим.

  – Прощающая. Другое важное качество успешных стратегий — уметь прощать. Отомстив, они должны вернуться к сотрудничеству, если оппонент не продолжает предавать. Это предотвращает бесконечное мщение друг другу и максимизирует выигрыш.

  – Независливая. Последнее качество — не быть завистливым, то есть не пытаться набрать больше очков, чем оппонент.

- Равновесие Нэша - концепция решения, одно из ключевых понятий теории игр. Так называется набор стратегий в игре для двух и более игроков, в котором ни один участник не может увеличить выигрыш, изменив свою стратегию, если другие участники своих стратегий не меняют. Джон Нэш доказал существование такого равновесия в смешанных стратегиях в любой конечной игре.

2.2 Реализация

Ниже приведен код на языке Haskell, запускающий игру с написанными ходами ползователя и выводит все совершенные ходы и результат для двух стратегий: «добрая» стратегия и равновесие по Нэшу.

```haskell
type IntTuple = (Int, Int)
type ListMatrix = [[IntTuple]]

-- Матрица очков для дилеммы заключенного
winningMatrix :: ListMatrix
winningMatrix = [[(3, 3), (0, 5)], [(5, 0), (1, 1)]]

-- Функция выбора стратегии по Нэшу
nashEquilibrium :: [Int] -> ListMatrix -> Int
nashEquilibrium _ winning =
    fst $ head [ (i, j) | i <- [0..1], j <- [0..1], isNash i j]
    where
        isNash i j = and [bestRow i j, bestCol i j]

        -- Проверка, что i выбор лучше в строке
        bestRow i j = all (\k -> fst (winning !! i !! j) >= fst (winning !! k !! j))
        ↪  [0..1]
```

```haskell

        -- Проверка, что j выбор лучше в колонке
        bestCol i j = all (\l -> snd (winning !! i !! j) >= snd (winning !! i !! l))
        ↪  [0..1]

-- Функция "доброй" стратегии
kindStrategy :: [Int] -> ListMatrix -> Int
kindStrategy prevMoves _
    | any (== 1) prevMoves = 1 -- Если оппонент когда-либо предал, начинаем предавать
    | otherwise = 0 -- Сотрудничаем до первого предательства

-- Основная логика игры
playGame :: ([Int] -> ListMatrix -> Int) -> ListMatrix -> [Int] -> [IntTuple]
playGame strategyF winningMatrix moves = results
    where
        computerMoves = map (\idx -> strategyF (take idx moves) winningMatrix)
        ↪  [1..length moves]
        results = zipWith (\playerMove cMove -> (playerMove, cMove)) moves computerMoves

scoreIncr :: IntTuple -> IntTuple -> IntTuple
scoreIncr (playerScore, computerScore) scores = (playerScore + fst scores, computerScore
↪  + snd scores)

-- Подсчет очков
calcScore :: [IntTuple] -> IntTuple -> IntTuple
calcScore [] (playerScore, computerScore) = (playerScore, computerScore)
calcScore ((pChoice, cChoice):results) (playerScore, computerScore)
    | (pChoice, cChoice) == (1, 1) = calcScore results $ scoreIncr (playerScore,
    ↪  computerScore) $ winningMatrix !! 0 !! 0
    | (pChoice, cChoice) == (1, 0) = calcScore results $ scoreIncr (playerScore,
    ↪  computerScore) $ winningMatrix !! 0 !! 1
    | (pChoice, cChoice) == (0, 1) = calcScore results $ scoreIncr (playerScore,
    ↪  computerScore) $ winningMatrix !! 1 !! 0
    | (pChoice, cChoice) == (0, 0) = calcScore results $ scoreIncr (playerScore,
    ↪  computerScore) $ winningMatrix !! 1 !! 1

-- Запуск игры с разными стратегиями
main :: IO ()
main = do
    -- 1 - предать, 0 - хранить молчание
    let playerMoves = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1] -- Список ходов игрока

    let nashRounds = playGame nashEquilibrium winningMatrix playerMoves
    let kindRounds = playGame kindStrategy winningMatrix playerMoves

    let nashScores = calcScore nashRounds (0, 0)
    let kindScores = calcScore kindRounds (0, 0)
    putStrLn "1 - betray, 0 - cooperate"
    putStrLn "Nash:"
    print nashRounds
```

```
60      putStrLn $ "Scores (Player, Computer): " ++ show nashScores
61
62      putStrLn "Kind:"
63      print kindRounds
64      putStrLn $ "Scores (Player, Computer): " ++ show kindScores
```

2.3 Результаты

В результате выполнения программы выводится список пар ходов игрока и компьютера, где на 1 месте в паре стоит ход игрока, на 2 - ход компьютера. Значение «1» обозначает предательство, «0» - сотрудничество (Рис. 3). После списка с ходами выводятся очки, где чем меньше число, тем лучше результат.

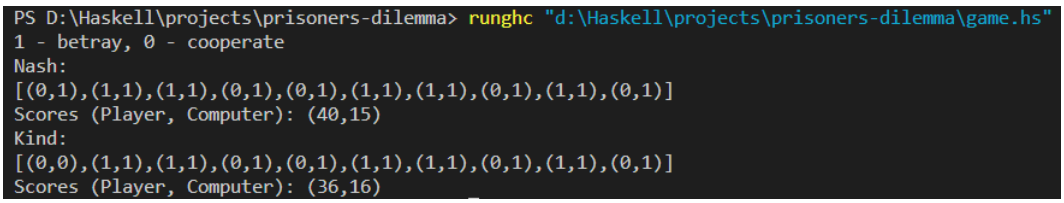

Рис. 3. Повторяющаяся дилемма заключенного

3 Заключение

В ходе выполнения практического задания были реализованы два проекта на языке программирования Haskell. Первый проект представлял собой реализацию фрактала кривой Пеано, который вычислял координаты всех точек на заданной глубине 'n' и формировал список списков пар координат для каждого уровня рекурсии. Второй проект включал в себя реализацию игры повторяющаяся дилемма заключенного, где была реализована стратегия равновесия по Нэшу и стратегия «добрая». Каждый проект был сохранен в отдельном `.hs` файле.