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»

Отчет по лабораторной работе №2
по дисциплине
«Функциональное программирование»
Вариант 3

Студент,
группы 5130201/20102 _____ Тищенко А. А.

Преподаватель,
к. т. н., доц. _____ Моторин Д. Е.

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

1 Математическое описание **3**

1.1 Фракталы . . . 3

1.2 Папоротник Барнсли . . . 3

1.3 Дилемма заключённого . . . 5

1.4 Равновесие Нэша . . . 5

1.5 Прощающая стратегия . . . 6

2 Особенности реализации **7**

2.1 Папоротник Барнсли . . . 7

2.1.1 Аффинные преобразование точек . . . 7

2.1.2 Генерация новой точки . . . 7

2.1.3 Рекурсивная генерация папоротника Барнсли . . . 8

2.2 Повторяющаяся дилемма заключённого . . . 8

2.2.1 Равновесие Нэша . . . 8

2.2.2 Прощающая стратегия . . . 9

2.2.3 Подсчёт очков . . . 9

2.2.4 Организация игры . . . 9

3 Результаты работы программы **11**

Заключение **12**

Список литературы **13**

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» [1]. 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:

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

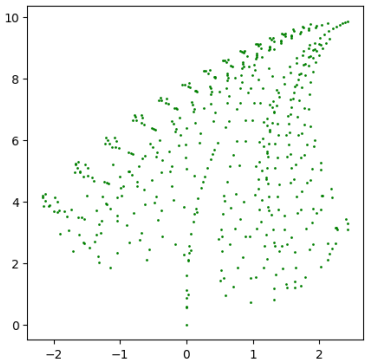

Рис. 1. Папоротник Барнсли для n = 500.

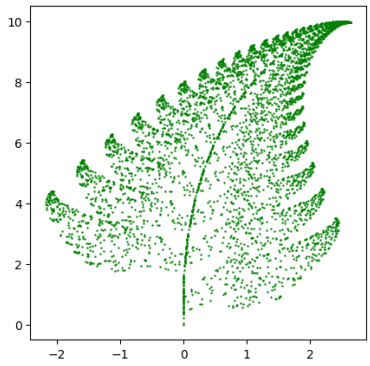

Рис. 2. Папоротник Барнсли для n = 5000.

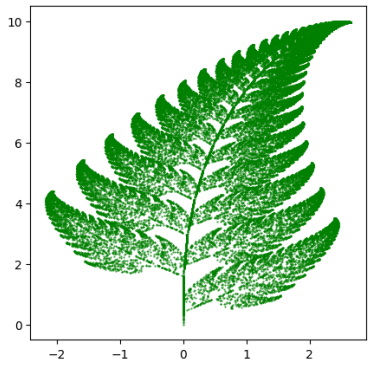

Рис. 3. Папоротник Барнсли для n = 50000.

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).

1.3 Дилемма заключённого

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 классическом варианте дилеммы заключённого два игрока могут выбрать одно из двух действий:

- **Сотрудничество**: Игрок решает сотрудничать с другим игроком, не сдавая его властям.
- **Предательство**: Игрок решает предать другого, сдавая его властям, с целью получения личной выгоды.

В зависимости от действий обоих игроков возможны следующие исходы:

- Если оба игрока сотрудничают, они оба получают умеренное наказание.
- Если один игрок сотрудничает, а другой предаёт, то предатель получает свободу, а сотрудничающий — максимальное наказание.
- Если оба игрока предают друг друга, оба получают умеренные наказания, однако хуже, чем при сотрудничестве.

В таблице 1 представлены варианты исходов дилеммы заключённого, которые рассматривались в этом варианте лабораторной работы.

Таблица 1. Исходы дилеммы заключённого: С — сотрудничество, П — предательство

	Игрок 2: С	**Игрок 2: П**
Игрок 1: С	1 год / 1 год	10 лет / 0 лет
Игрок 1: П	0 лет / 10 лет	5 лет / 5 лет

В повторяющейся дилемме заключённого игра происходит периодически, и каждый игрок может «наказать» другого за несотрудничество ранее.

1.4 Равновесие Нэша

Равновесие Нэша — 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.

Определить стратегию, при которой наступает равновесие можно с помощью формулы:

$$u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i}) \quad \text{для всех } s_i,$$

где:

- $u_i(s_i^*, s_{-i})$ — выигрыш игрока i, если он выбирает стратегию s_i^*, а остальные игроки выбирают стратегии s_{-i};
- s_{-i} — набор стратегий всех игроков, кроме игрока i;
- s_i — стратегия игрока i.

В случае с дилеммой заключённого равновесие Нэша достигается, когда оба игрока выбирают стратегию предательства (П). Если игрок 1 выбирает сотрудничество (С), то игрок 2, выбирая предательство (П), получает 0 лет (лучший результат для него), а игрок 1 — 10 лет. Если игрок 1 выбирает предательство (П), то игрок 2, также выбирая предательство (П), получает 5 лет, а игрок 1 — 5 лет.

1.5 Прощающая стратегия

Прощающая стратегия заключается в том, что игрок не предает, пока это не сделает оппонент, далее мстит один ход и снова возвращается к сотрудничеству, если оппонент не продолжает предавать.

$$s_i(t) = \begin{cases} \text{С}, & \text{если } s_j(t-1) = \text{С}, \\ \text{П}, & \text{если } s_j(t-1) = \text{П}. \end{cases}$$

где:

- $s_i(t)$ — стратегия игрока i на шаге t,
- $s_j(t-1)$ — стратегия оппонента на шаге $t-1$.

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. Код функций, резализующих аффинные преобразования над точками для построения папоротника Барнсли.

```
1   type Point = (Float, Float)
2
3   transformation1 :: Point -> Point
4   transformation1 (_, y) = (0, 0.16 * y)
5
6   transformation2 :: Point -> Point
7   transformation2 (x, y) = (0.85 * x + 0.04 * y, -0.04 * x + 0.85 * y + 1.6)
8
9   transformation3 :: Point -> Point
10  transformation3 (x, y) = (0.2 * x - 0.26 * y, 0.23 * x + 0.22 * y + 1.6)
11
12  transformation4 :: Point -> Point
13  transformation4 (x, y) = (-0.15 * x + 0.28 * y, 0.26 * x + 0.24 * y + 0.44)
14
```

2.1.2 Генерация новой точки

Генерация новой точки происходит с помощью функции `genNextPoint` и вспомогательной функции `applyTransformation`, код которых представлен в листинге 2. `applyTransformation` 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. `genNextPoint` принимает на вход исходную точку и состояние генератора случайных чисел – `gen`, 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.

```
1   import System.Random (StdGen, mkStdGen, randomR)
2
3   applyTransformation :: Point -> Float -> Point
4   applyTransformation point random
5       | random < 0.01 = transformation1 point
6       | random < 0.86 = transformation2 point
7       | random < 0.93 = transformation3 point
8       | otherwise = transformation4 point
9
```

```
10    genNextPoint :: StdGen -> Point -> (Point, StdGen)
11    genNextPoint gen point =
12        let (random, newGen) = randomR (0.0, 1.0 :: Float) gen
13        in (applyTransformation point random, 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.

```
1     barnsleyFern :: StdGen -> Point -> Int -> [Point]
2     barnsleyFern _ _ 0 = []
3     barnsleyFern gen startPoint n =
4         let (nextPoint, newGen) = genNextPoint gen startPoint
5         in startPoint : barnsleyFern newGen nextPoint (n - 1)
```

2.2 Повторяющаяся дилемма заключённого

2.2.1 Равновесие Нэша

Функция `nashEquilibriumStrategy`, код которой представлен в листинге 4, реализует равновесие Нэша. Функция рекурсивная, она принимает на вход список действий оппонента, список уже сгенерированных действий и число – счётчик количества ходов. Функция завершается, когда достигается максимум ходов, либо когда заканчиваются ходы оппонента. Она возвращает список ходов игрока в соответствии со стратегией.

Листинг 4. Код функции, резализующей стратегию в соответствии с равновесием Нэша.

```
1     nashEquilibriumStrategy :: [Char] -> [Char] -> Int -> [Char]
2     nashEquilibriumStrategy opponentMoves generatedMoves n =
3     if n <= 100 && length opponentMoves > 0
4         then
5         nashEquilibriumStrategy (tail opponentMoves) (generatedMoves ++ [nextStep]) (n + 1)
6         else generatedMoves
7     where
8         cases = [[' С', ' С'], [' С', ' П'], [' П', ' С'], [' П', ' П']]
9         results = [[1, 1], [10, 0], [0, 10], [5, 5]]
10        p_years = min (results !! 1 !! 1) (results !! 3 !! 1)
11        s_years = min (results !! 0 !! 1) (results !! 2 !! 1)
12        nextStep | p_years <= s_years = ' П'
13                 | otherwise = ' С'
14
```

2.2.2 Прощающая стратегия

Функция `forgivingStrategy`, код которой представлен в листинге 5, реализует прощающую стратегию. Функция рекурсивная, она принимает на вход список действий оппонента, список уже сгенерированных действий и число – счётчик количества ходов. Функция завершается, когда достигается максимум ходов, либо когда заканчиваются ходы оппонента. Она возвращает список ходов игрока в соответствии со стратегией.

Листинг 5. Код функции, реализующей прощающую стратегию.

```
forgivingStrategy :: [Char] -> [Char] -> Int -> [Char]
forgivingStrategy opponentMoves generatedMoves n
    | n > 100 || length opponentMoves == 1 = generatedMoves
    | n == 0
        = forgivingStrategy opponentMoves (generatedMoves ++ C['']) (n + 1)
    | head opponentMoves == C''
        = forgivingStrategy (tail opponentMoves) (generatedMoves ++ C['']) (n + 1)
    | otherwise
        = forgivingStrategy (tail opponentMoves) (generatedMoves ++ П['']) (n + 1)

```

2.2.3 Подсчёт очков

Функция `getScore`, код которой представлен в листинге 6, подсчитывает количество лет заключения для каждого игрока. Функция рекурсивная, она принимает на вход список действий первого игрока, список действий второго игрока, и два числа – количество лет заключения для перового и второго игроков. Функция возвращает пару из двух чисел – количества лет заключения для игроков.

Листинг 6. Код функции для подсчёта лет заключения.

```
getScore :: [Char] -> [Char] -> Int -> Int -> (Int, Int)
getScore moves1 moves2 score1 score2 =
    if length moves1 == 0 then (score1, score2)
    else getScore (tail moves1) (tail moves2) newScore1 newScore2
    where
        cases = [[' C', ' C'], [' C', ' П'], [' П', ' C'], [' П', ' П']]
        results = [[1, 1], [10, 0], [0, 10], [5, 5]]
        newScore1 = score1 + results !! indexOf cases [head moves1, head moves2] !! 0
        newScore2 = score2 + results !! indexOf cases [head moves1, head moves2] !! 1

```

2.2.4 Организация игры

Функция `game`, код которой представлен в листинге 7, запускает игру. Она принимает на вход список действий первого игрока и функцию, реализующую стратегию игры. Функция возвращает список ответных ходов, выбранных в соответствии с указанной стратегией.

Листинг 7. Код функции для подсчёта лет заключения.

```
1    game :: [Char] -> ([Char] -> [Char] -> Int -> [Char]) -> [Char]
2    game playerMoves gameStrategy
3        | null playerMoves = []
4        | otherwise = gameStrategy playerMoves [] 0
5
```

3 Результаты работы программы

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

```
Укажите количество шагов рекурсии:
20
[(0.0,0.0),(0.0,1.6),(6.4e-2,2.96),(0.1728,4.11344),(0.31141758,5.0895123),(0.46828544,5.913
6286),(0.63458776,6.607853),(0.80371374,7.191292),(0.9708083,7.68045),(1.132405,8.089551),(1
.2861264,8.430822),(1.4304404,8.714754),(1.5644646,8.950324),(1.6878078,9.145197),(1.8004446
,9.305905),(1.9026141,9.438003),(-2.0733578,4.1139617),(-1.5977957,5.179802),(-1.1509343,6.0
667434),(-0.73562443,6.802769)]
```

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

```
Ходы компьютера: ССПССППССП
Годы заключения: 33 (игрок) - 43 (компьютер)
```

Рис. 5. Повторяющаяся дилемма заключённого при прощающей стратегии.

```
Ходы компьютера: ППППППППП
Годы заключения: 75 (игрок) - 25 (компьютер)
```

Рис. 6. Повторяющаяся дилемма заключённого для равновесия по Нэшу.

На Рис. 5 представлена повторяющаяся дилемма заключённого для прощающей стретегии при заданных ходах игрока: `['С', 'П', 'С', 'С', 'П', 'П', 'С', 'С', 'П', 'П']`. А на Рис. 6 для равновесия по Нэшу.

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, генерирует действия второго заключённого при заданной стратегии и действиях первого заключённого. В рамках лабораторной работы реализованы две стратегии: прощающая стратегия и стратегия основанная на равновесии по Нэшу.

Список литературы

[1] Michael F. Barnsley, Hawley Rising. Fractals Everywhere. — Morgan Kaufmann, 1993-01-01. — 568 с.

[2] Колокольцов, В. Н., Малафеев, О. А. «Математическое моделирование многоагентных систем конкуренции и кооперации (Теория игр для всех)». — Санкт-Петербург: Издательство Лань, 2022. — 624 с.