МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО"
Институт компьютерных наук и технологий
Направление **02.03.01** : Математика и компьютерные науки

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОГО ЗАДАНИЯ 2

«Введение в Haskell»

Исполнитель: _____ Яшнова Дарья Михайловна
группа 5130201/20002

Руководитель: _____ Моторин Дмитрий Евгеньевич

«___»_____2024г

Санкт-Петербург, 2024

Введение

Данное практическое задание посвящено введению в функциональное программирование на языке Haskell. Задача заключается в реализации фрактала "Дерево Пифагора"и симуляции игры «НИМ».

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

Реализация игры «НИМ» включает в себя функцию, организующей игру (игра должна продолжаться не более 100 ходов). Игра представляет собой «НИМ» - (N кучек в каждой не более M камней), где игрок может взять неограниченное число камней.

Для каждой части задания необходимо разработать .hs-файл, содержащий весь необходимый код на языке Haskell.

Содержание

1 Задача 1: Фрактал «Дерево Пифагора»

1.1 Общие сведения о фракталах

Фрактал — это фигура, обладающая свойством самоподобия. Объект называют самоподобным, если одна или более его частей похожа на его целое. При этом количество повторяющихся частей у фрактала стремится к бесконечности — этим он отличается от самоподобных геометрических фигур с конечным числом звеньев (предфракталов).

Фракталы принято делить на геометрические, алгебраические и стохастические.

1.2 Фрактал «Дерево Пифагора»

Пифагор, доказывая свою знаменитую теорему, построил фигуру, где на сторонах прямоугольного треугольника расположены квадраты. Впервые дерево Пифагора построил А.Е.Босман (1891—1961) во время Второй мировой войны, используя обычную чертёжную линейку. Одним из свойств дерева Пифагора является то, что если площадь первого квадрата равна единице, то на каждом уровне сумма площадей квадратов тоже будет равна единице.

Если в классическом дереве Пифагора угол равен 45 градусам, то также можно построить и обобщённое дерево Пифагора при использовании других углов. Такое дерево часто называют обдуваемое ветром дерево Пифагора.

1.3 Вычисление координат квадратов

В реализации фрактала для задания каждого квадрата используются 2 точки - центр квадрата и точка, являющаяся серединой верхней стороны квадрата. В каждой итерации верхняя сторона квадрата - это та сторона, на которой строится треугольник и квадраты следующей итерации.

Пусть (x_n, y_n) координаты центра квадрата, (u_n, v_n) - координаты середины верхней стороны квадрата на n-ной итерации. Тогда:

$$\overrightarrow{V} = (u_n - x_n, v_n - y_n);$$

$$\overrightarrow{V}_{\perp L} = (v_n - y_n, x_n - u_n);$$

$$\overrightarrow{V}_{\perp R} = (y_n - v_n, u_n - x_n).$$

Тогда, как можно увидеть из рис.1, координаты точек двух квадратов на n+1-ой итерации можно выразить так:

$$(x^1_{n+1}, y^1_{n+1}) = (u, v) + \overrightarrow{V} + \overrightarrow{V}_{\perp R}$$

$$(u^1_{n+1}, v^1_{n+1}) = (x^1_{n+1}, y^1_{n+1}) + \frac{\overrightarrow{V}}{2} + \frac{\overrightarrow{V}_{\perp R}}{2}$$

$$(x^2_{n+1}, y^2_{n+1}) = (u, v) + \overrightarrow{V} + \overrightarrow{V}_{\perp L}$$

$$(u^2_{n+1}, v^2_{n+1}) = (x^2_{n+1}, y^2_{n+1}) + \frac{\overrightarrow{V}}{2} + \frac{\overrightarrow{V}_{\perp L}}{2}$$

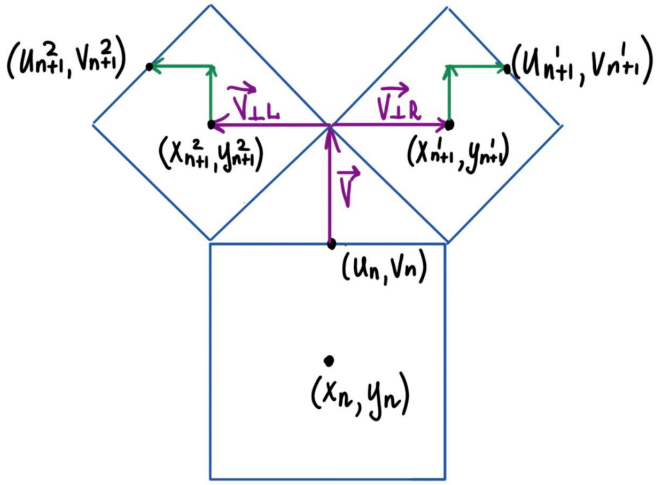

Рис. 1: Визуализация вычисления n+1-ой итерации после n-ой

На рис.2 представлена визуализация первх 4 шагов алгоритма построения.

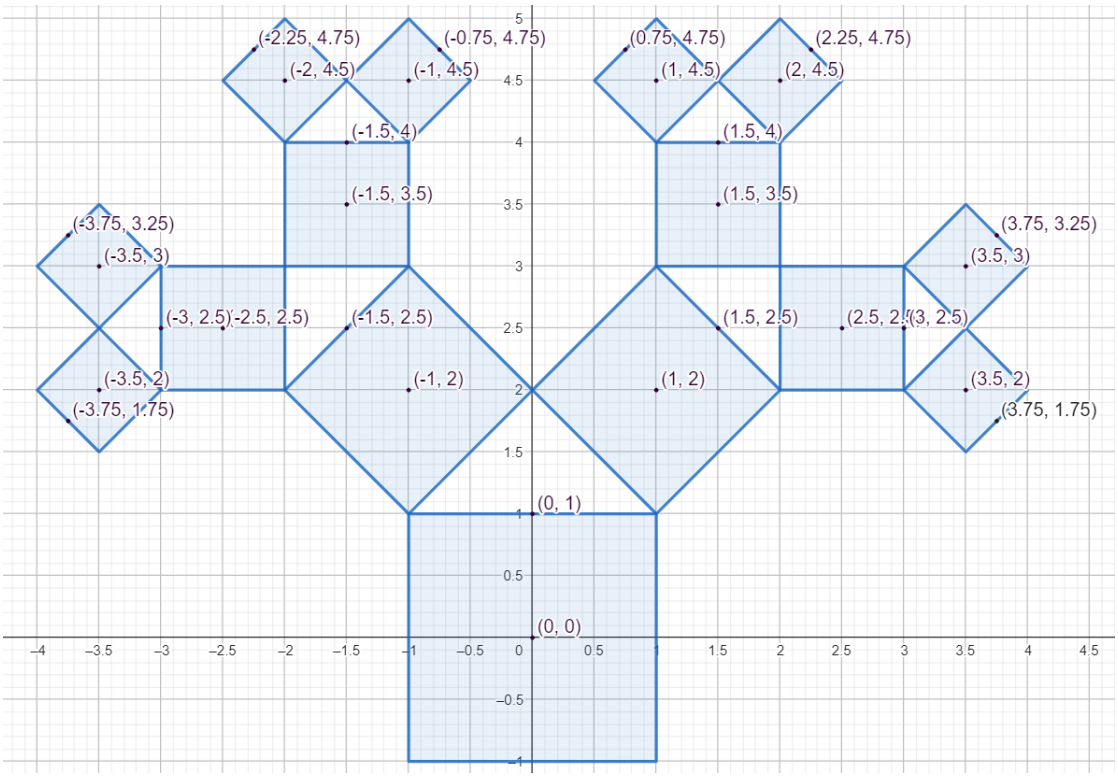

Рис. 2: Визуализация первых 4 шагов построения дерева Пифагора

Квадрат однозначно восстанавливается по двум заданным точкам. Достаточно соединить их - длина отрезка будет половиной длины стороны. Если построить через вторую точку перпендикуляр к этому отрезку длинной в 2 раза больше, так, чтобы эта точка была серединой, то получится одна из сторон квадрата.

1.4 Код программы и результат работы программы

Код программы состоит из нескольких частей:

1. Определение типа данных Square:

   - data Square = Square (Double, Double) (Double, Double) определяет тип данных Square для представления квадрата. Каждый квадрат определяется двумя точками (x, y) и (u, v) в двумерном пространстве.

   - instance Show Square определяет функцию для преобразования Square в строку, что позволяет выводить его на экран.

2. Функция fillSquare:

   - fillSquare :: Double -> Double -> Double -> Double -> Square принимает четыре числа (x, y, u, v) и создает объект Square с этими координатами.

3. Константа zeroSquare:

   - zeroSquare = fillSquare 0.0 0.0 0.0 1.0 создает начальный квадрат с координатами (0, 0), (0, 1).

4. Функции leftSquare и rightSquare:

   - leftSquare (Square (x ,y) (u ,v)) и rightSquare - это функции, которые определяют рекурсивный алгоритм построения Дерева Пифагора. Они принимают квадрат и возвращают два новых квадрата - «левый» и «правый», которые образуют следующую итерацию фрактала.

5. Функция fractal:

   - fractal :: Int -> Square -> [Square] рекурсивно строит Дерево Пифагора, начиная с заданного квадрата и повторяя построение "левого"и "правого"квадратов n раз. Результат - список всех квадратов, составляющих фрактал.

6. Функция grFrac:

   - grFrac n square - функция формирует список списков квадратов, представляющих Дерево Пифагора на разных уровнях рекурсии.

   При работе программы пользователю выводится на экран список списков координат квадратов (рис.3).

```
ghci> grFrac 2 zeroSquare
[[(-2.5,2.5)(-3.0,2.5),(-1.5,3.5)(-1.5,4.0),(1.5,3.5)(1.5,4.0)
,(2.5,2.5)(3.0,2.5)],[(-1.0,2.0)(-1.5,2.5),(1.0,2.0)(1.5,2.5)]
,[(0.0,0.0)(0.0,1.0)]]
```

Рис. 3: Пример работы программы

Первая координата в выводе - координата центра квадрата, вторая - координата середины одной из сторон. Для n-ого уровня выведется 2^n квадратов.

2 Задача 2: Игра НиМ

2.1 Теоретические сведения об игре НиМ

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 трём.

Спецификация задания состоит в том что, игра должна идти с n кучками, в каждой из которых не более m корней, игроки могут взять любое количество корней. Вся игра должна продолжаться не более 100 ходов.

2.2 Реализация стратегии

В данной игре реализовано 2 стратегии. «Случайная» стратегия в каждом состоянии кучек выбирает случайное число камней в интервале между максимальным количеством камней в одной кучке и минимальным. Если H_i - количество камней в i-той кучке, то будет сделан ход

$$X = rand\_number(min_{i=0}^{i=n-1} H_i, min_{i=0}^{i=n-1} H_i)$$

из кучки с максимальным количеством камней.

Также реализована стратегия, сохраняющая НиМ-сумму. НиМ-сумма - результат побитового «исключающего или». Пример поиска НиМ-суммы:

$$H_0 = 00101_2$$

$$H_1 = 00100_2$$

$$H_2 = 10100_2$$

$$H_3 = 00110_2$$

$$H_4 = 00110_2$$

НиМ-сумма: 10101_2

Для того, чтобы сохранять НиМ-сумму равной 0, вычисляется текущая НиМ-сумма: $A = \oplus_{i=0}^{n-1} H_i$. Чтобы понять сколько камней мы будем брать, нужно применить побитовое исключающее или к кучке с максимальным количеством камней и текущей НиМ-сумме:

$$X = A \oplus H^{max}.$$

В этот ход должно быть взято X камней из кучки H^{max}. Тогда НиМ-сумма станет равна 0. Заметим, что, если перед этим ходом A = 0, то невозможно следующим ходом сделать ее равной 0. Неважно из какой кучки брать камни, независимо от их количества в НиМ-сумме появятся единицы. Поэтому соперник не сможет после этого хода сделать НиМ-сумму равной 0. В любом другом случае сделать НиМ-сумму равной 0 возможно. Значит, после хода соперника, всегда можно сходить соответствуя стратегии.

2.3 Код реализации

Код реализации программы состоит из нескольких частей:

1. Импорт модулей: Импортируются модули System.Random, Data.Bits, Data.List, Control.Monad, System.IO для работы со случайными числами, битовыми операциями, списками, монадным программированием, вводом-выводом.

2. Функции для игры:

   (a) heaps n m: Создает начальное состояние игры - список куч с случайным количеством камней в каждой.

   (b) heapSum: Считает общее количество камней во всех кучах.

   (c) genChoice (x:xs): Генерирует случайное число камней для удаления из кучи.

   (d) findAndReduce: Удаляет заданное количество камней из заданной кучи в списке куч.

   (e) randStrategy: Генерирует случайный ход (удаление камней из случайной кучи).

   (f) nimSum: Вычисляет Ним-сум - значение, которое помогает определить оптимальный ход.

   (g) replaceFirst: Заменяет первый элемент в списке на новый.

   (h) xorStrategy: Определяет оптимальный ход, используя XOR (исключающее ИЛИ) и Ним-сум.

   (i) decreaseIthElement: Уменьшает количество камней в указанной куче.

   (j) userChoice: Взаимодействует с пользователем, получает от него ход и проверяет его корректность.

   (k) gameUtil: Рекурсивно управляет ходами игры, чередуя ходы пользователя и компьютера (стратегия XOR), выводит на экран текущее состояние игры и объявляет победителя.

   (l) game: Инициализирует игру, задавая количество куч и камней в каждой.

3. Игровой цикл:

   (a) Инициализируется начальное состояние игры.

   (b) В цикле gameUtil рекурсивно чередуются ходы пользователя и компьютера до тех пор, пока не будет достигнуто победное состояние или ограничение по количеству ходов.

   (c) В случае победы пользователя или компьютера, выводится сообщение о победителе.

   (d) В случае ничьей, выводится сообщение о ничьей и текущее состояние игры.

На рис.4-5 представлен вывод на экран в процессе игры.

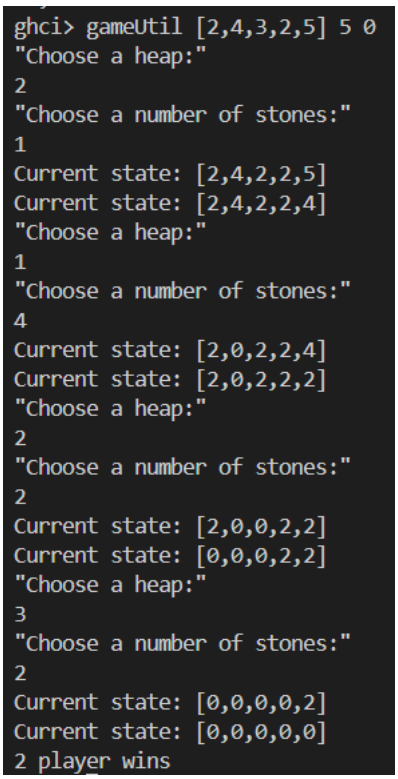

Рис. 4: Вывод на экран в процессе игры с заданными вручную кучами

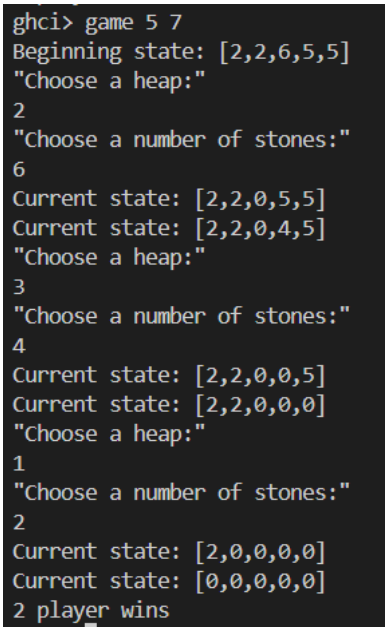

Рис. 5: Вывод на экран в процессе игры с случайно генерируемыми кучами

В первом случае (рис.4) пользователь сам задает размеры кучек, а во втором (рис.5) выбирает максимальное число камней в куче и количество кучек. Сами кучки генерируются случайно.

Выводы

В рамках задания были разработаны две программы: программа для построения фрактала «Дерево Пифагора» и программа для симуляции игры НиМ.

Фрактал был реализован рекурсивно, генерируя пары координат для каждого уровня рекурсии. Результат - список списков пар координат, где каждый список представляет собой уровень фрактала. Визуализация фрактала выполнена с использованием стороннего редактора Geogebra.

Игра НиМ была реализована с использованием стратегии XOR для определения оптимального хода компьютера. Программа позволяет пользователю совершать ходы, а затем демонстрирует действия компьютера. Результатом игры является определение победителя или объявление ничьей.

Задание выполнено в среде Visual Studio Code, GHCi, версия 9.4.8.

Список литературы

1. «Machines designed to play Nim games. Teaching supports for mathematics, algorithmics and computer science», https://hal.science/hal-01349260/document , Lisa Rougetet(дата обращения 18.10.2024)

2. «Novel Modified Pythagorean Tree Fractal Monopole Antennas for UWB Applications», https://www.researchgate.net/publication/224237056_Novel_Modified_Pythagorean_Tree_Fractal_Monopole_Antennas_for_UWB_Applications, Javad Pourahmadazar (дата обращения 24.10.2024)

Приложение А. Код реализации фрактала

```
1       import Control.Monad (forM_)
2
3   data Square = Square (Double, Double) (Double, Double)
4   instance Show Square where
5       show (Square (x ,y) (u ,v) ) = "(" ++ show x ++","++ show y ++")("++
            show u ++","++ show v++")"
6   -- Функция для заполнения кортежа
7   fillSquare :: Double -> Double -> Double -> Double -> Square
8   fillSquare x y u v = Square (x, y) (u, v)
9
10  zeroSquare :: Square
11  zeroSquare = fillSquare 0.0 0.0 0.0 1.0
12
13  -- Доступ к элементам кортежа
14  getX :: Square -> Double
15  getX (Square (x, _) (_ ,_)) = x
16
17  getY :: Square -> Double
18  getY (Square(_ ,y)(_, _)) = y
19
20  getU :: Square -> Double
21  getU (Square (_ ,_ )(u, _)) = u
22
23  getV :: Square -> Double
24  getV (Square (_, _) (_, v)) = v
25
26  printSq (Square (x ,y) (u ,v)) = do
27       print ( "{" ++ show x ++", "++ show y ++"), ("++show u ++", "++ show
             v++"} ")
28
29  leftSquare:: Square -> Square
30  leftSquare (Square (x ,y) (u ,v)) = fillSquare (2*u-x+y-v)  (2*v-y+u-x)
       (2*u-x+y-v+(u-x)/2+(y-v)/2)  (2*v-y+u-x+(u-x)/2+(v-y)/2)
31
32  rightSquare:: Square -> Square
33  rightSquare (Square (x ,y) (u ,v)) = fillSquare (2*u-x+v-y)  (2*v-y+x-u)
       (2*u-x+v-y+(u-x)/2+(v-y)/2)  (2*v-y+x-u+(v-y)/2+(x-u)/2)
34
35  fractal :: Int -> Square -> [Square]
36  fractal 0 square = [square]
37  fractal n square = fractal (n - 1) (leftSquare square) ++ fractal (n - 1)
       (rightSquare square)
38
39  grFrac (0) square = [[square]]
40  grFrac n square = [fractal n square] ++  grFrac (n-1) square
```

Приложение Б. Код реализации игры НиМ

```
1       import System.Random
2   import Data.Bits
3   import Data.List
4   import Control.Monad
5   import System.IO
6   import GHC.Base (VecElem(Int16ElemRep))
7   import System.Posix.Internals (lstat)
```

```
-- 1

-- game player1 player2 n m
heaps n m = replicateM n (randomRIO (1, m))


heapSum (x:xs) = sum(x:xs)


genChoice (x:xs) = k
 where k = randomRIO (max(minimum (x:xs)) 1, maximum (x:xs))

findAndReduce :: (Ord t, Num t) => t -> [t] -> [t]
findAndReduce k [] = []
findAndReduce k (x:xs)
  | x >= k     = (x - k) : xs
  | otherwise = x : findAndReduce k xs

randStrategy :: [Int] -> IO [Int]
randStrategy (x:xs) = do
  k <- genChoice (x:xs)
  return $ findAndReduce k (x:xs)
nimSum (x:xs) = foldl xor x xs

replaceFirst oldElement newElement list =
 case findIndex (== oldElement) list of
  Nothing -> list
  Just i -> take i list ++ [newElement] ++ drop (i + 1) list

xorStrategy xs = do
     let s = nimSum xs
     let a = maximum xs
     let updA = min (xor a s) (a-1)
     return (replaceFirst a updA xs)

decreaseIthElement i amount xs =
  if i < 0 || i >= length xs
  then xs -- Return original list if i is invalid
  else take i xs ++ [xs !! i - amount] ++ drop (i + 1) xs

userChoice :: [Int] -> Int -> IO [Int]
userChoice (x:xs) n = do
     print "Choose a heap:"
     heap<-getLine
     let heap_ = read heap::Int
     print "Choose a number of stones:"
     stones<-getLine
     let stones_ = read stones::Int
     if heap_>n-1 || (x:xs) !! heap_ <stones_
     then do
          print "Wrong enter. Try again"
          userChoice (x:xs) n
     else do
          let lst = decreaseIthElement heap_ stones_ (x:xs)
          putStrLn $ "Current state: " ++ show lst -- Print the list
          return lst

gameUtil :: Integral t => [Int] -> Int -> t -> IO ()
gameUtil xs n it = if it > 100
     then putStrLn $ "It is draw. Current state: " ++ show xs
```

```
68        else if heapSum xs == 0
69             then
70             if mod it 2 == 0
71             then putStrLn $ "2 player wins"
72             else putStrLn $ "1 player wins"
73        else do
74             if mod it 2 == 0
75             then do
76                  lst <- userChoice xs n
77                  gameUtil lst n (it + 1)
78             else do
79                  lst <- xorStrategy xs
80                  putStrLn $ "Current state: " ++ show lst
81                  gameUtil lst n (it + 1)
82
83
84  game :: Int -> Int -> IO ()
85  game n m  = do
86        lst <- (heaps n m)
87        putStrLn $ "Beginning state: " ++ show lst
88        gameUtil lst n 0
```