МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ **(ИТПИ)**
Кафедра программной инженерии и вычислительной техники **(ПИиВТ)**

.

Дисциплина: «Разработка приложений искусственного интеллекта в киберфизических системах»
Лабораторная работа №5.

Тема: «Определение координат точек пересечения окружности со сторонами ромба»

Выполнил:

Студент группы ИКПИ-23

Харлова А.А

Подпись _____

Принял:

Ерофеев С.А.

Подпись _____

2024 г.

Постановка задачи

Написать программу на языке функционального программирования Haskell, которая определяет точки пересечения окружности со сторонами ромба. Окружность задаётся координатами центра и радиусом, ромб - четырьмя вершинами.

Описание программы

Задача определения координат точек пересечения окружности со сторонами ромба подразумевает аналитическое решение геометрической задачи.

В ходе выполнения работы необходимо решить систему уравнений, содержащую уравнение окружности и прямых, на которых лежат заданные отрезки (стороны ромба), предварительно выведя их по известным координатам. Алгоритм решения кратко представлен на рисунке 1.

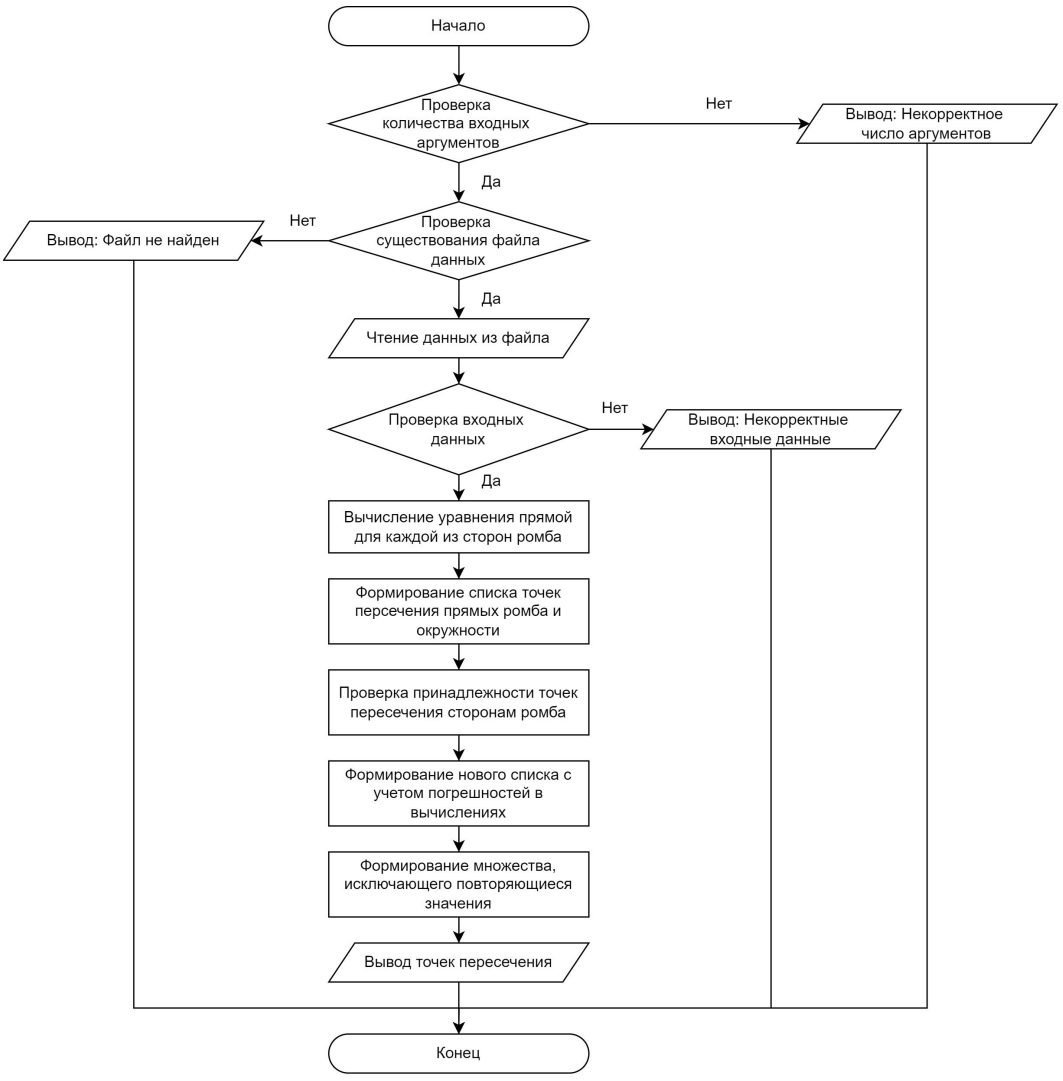

Рис. 1. Алгоритм решения задачи.

Перед началом основной работы программа должна считывать входные данные из файла, проверять корректность введенных координат: ромб должен быть задан так, чтобы его точки образовывали ромб, а не произвольный четырехугольник; окружность должна иметь положительный радиус. Вывод данных осуществляется в консоль.

В таблице 1 приведено описание функций, используемых в программе.

Функция	Описание
main	Главная функция программы. Обработка аргументов командной строки (имя файла)
mainProgramm	Основная логика программы
showInput	Вывод входных данных (координаты ромба и окружности)
showAnswer	Вывод результата (точки пересечения окружности с ромбом)
checkInput	Проверка корректности входных данных
correctRhombus	Упорядочивание точек ромба
roundTo	Округление координат точек до заданной точности
coeffs	Вычисление коэффициентов уравнения прямой по двум точкам
centre	Нахождение центра отрезка по координатам его концов
solveQuadratic	Решение квадратного уравнения
lineEquation	Вычисление y по уравнению прямой
intersection	Нахождение точки пересечения прямой и окружности
isPointOnSegment	Проверка, лежит ли точка на отрезке
isPointOnRhombus	Проверка, лежит ли точка на сторонах ромба
distance	Расстояние между двумя точками

Таблица 1. Используемые функции

Используемые формулы

В ходе разработки программы были использованы математические формулы, на основе которых было выведено уравнение для решения задачи. Все применённые формулы представлены в таблице 2.

Формула	Описание
$gy + kx + b = 0$	Общий вид уравнения прямой
$y = kx + b$	Уравнение наклонной прямой
$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}, x_2 \neq x_1$	Уравнение прямой, заданное двумя точками
$r^2 = (x - x_0)^2 + (y - y_0)^2$	Уравнение окружности
$Ax^2 + Bx + C = 0$	Формула квадратного уравнения
$D = B^2 - 4AC$	Формула нахождения дискриминанта
$x_{1,2} = \frac{-B \pm \sqrt{D}}{2A}, D >= 0$	Формула нахождения корней квадратного уравнения через дискриминант
$((\frac{k}{g})^2 + 1) \cdot x^2 + (2k \cdot \frac{b + y_0 * g}{g^2} - 2x_0) \cdot x + + (\frac{b}{g} + y_0)^2 + {x_0}^2 - r^2 = 0, g \neq 0$	Выведенная формула для определения координат пересечения окружности с горизонтальной или наклонной прямой
$y^2 - 2y_0 \cdot y + {y_0}^2 + (b - x_0)^2 - r^2 = 0$	Выведенная формула для определения координат пересечения окружности с вертикальной прямой

Таблица 2. Используемые формулы

Вывод

Была разработана программа на языке функционального программирования Haskell, определяющая точки пересечения окружности со сторонами ромба. В ходе разработки были введены необходимые предикаты. Выполнено тестирование. Поставленная задача выполнена.

Приложение

```haskell
module Main where
import System.Environment (getArgs)
import System.Directory (doesFileExist)
import System.IO (hPutStrLn, stderr)

import System.Exit (die)
import Data.List (nub)
import Data.Maybe (isJust, isNothing, fromJust)
import Control.Concurrent.STM (check)


-- Типы данных
type Point = (Double, Double)
type Line = (Point, Point)
type Circle = (Point, Double)

-- Точка входа
main :: IO ()
main = do
```

```haskell
    -- Получаем аргументы командной строки
    args <- getArgs

    -- Проверяем, что передан ровно один аргумент (имя файла)
    if length args /= 1
        then die "\27[31m> [ Error ]\27[0m: Incorrect arguments."
        else do
            -- Извлекаем имя файла из аргументов
            let fileName = head args
            fileExists <- doesFileExist fileName
            if not fileExists
                then die "\27[31m> [ Error ]\27[0m: File does not exist."
            else do
                mainProgramm fileName

-- Основная программа
mainProgramm :: FilePath -> IO ()
mainProgramm fileName = do
    input <- readFile fileName
    let [line1, line2, line3] = lines input
        rhombusInputed = read line1 :: [Point]
        circleCentre = read line2 :: Point
        circleRadius = read line3 :: Double
        circle = (circleCentre, circleRadius)
        rhombus = correctRhombus rhombusInputed

    if isJust rhombus
        then showInput (fromJust rhombus, circle)
        else showInput (rhombusInputed, circle)
    checkInput (rhombus, circleRadius)
    let
        [rhPoint1, rhPoint2, rhPoint3, rhPoint4] = fromJust rhombus
        rhombusLines = [coeffs rhPoint1 rhPoint2, coeffs rhPoint2 rhPoint3, coeffs
rhPoint3 rhPoint4, coeffs rhPoint4 rhPoint1]
        intersections = concatMap (\i -> case intersection (rhombusLines !! i)
(circleCentre, circleRadius) of
                                    Just ((x1, y1), (x2, y2)) -> [(x1, y1), (x2, y2)]
                                    Nothing -> []) [0..3]
        intersInRhombus = filter (`isPointOnRhombus` fromJust rhombus) intersections
        roundedIntersections = map (roundTo 10) intersInRhombus
    showAnswer . nub $ roundedIntersections

--Округление координат точки до определённой точности
roundTo n (x, y) = (rx, ry)
    where rx = (fromIntegral (round (x * (10^^n))) :: Double) / (10^^n)
          ry = (fromIntegral (round (y * (10^^n))) :: Double) / (10^^n)

--Нахождение коэффициентов уравнения прямой
coeffs :: Point -> Point -> (Double, Double, Double)
coeffs (x1, y1) (x2, y2)
    | y1 == y2 = (0, -1, min y1 y2)
    | y1 /= y2 && x1 /= x2 = ((y2 - y1)/(x2 - x1), -1, y1 - ((y2 - y1)/(x2 - x1))*x1)
    | y1 /= y2 && x1 == x2 = (-1,  0, min x1 x2)

-- Функция для проверки входных данных
checkInput :: (Maybe [Point], Double) -> IO ()
checkInput (rhombus, circleRadius)
    | isNothing rhombus = die "\27[31m> [ Error ]\27[0m: The entered figure is not a
rhombus."
    | circleRadius <= 0 = die "\27[31m> [ Error ]\27[0m: Circle radius must be
positive."
    | otherwise = putStrLn "\27[32m> Input is valid\27[0m"
```

```haskell
-- Упорядочивание введённых координат ромба
correctRhombus :: [Point] -> Maybe [Point]
correctRhombus [point1, point2, point3, point4]
    | centre point1 point3 == centre point2 point4 && side12 == side34 && side23 ==
side41 && side12 == side23 && side12 /= 0 && side24 /= 0 && side13 /= 0 && side23 /=
0 && side34 /= 0 && side41 /= 0 = Just [point1, point2, point3, point4]
    | centre point1 point2 == centre point3 point4 && side13 == side24 && side23 ==
side41 && side13 == side23 && side12 /= 0 && side24 /= 0 && side13 /= 0 && side23 /=
0 && side34 /= 0 && side41 /= 0 = Just [point1, point3, point2, point4]
    | centre point1 point4 == centre point2 point3 && side12 == side34 && side24 ==
side13 && side12 == side24 && side12 /= 0 && side24 /= 0 && side13 /= 0 && side23 /=
0 && side34 /= 0 && side41 /= 0 = Just [point1, point2, point4, point3]
    | otherwise = Nothing
    where
        side12 = distance point1 point2
        side24 = distance point2 point4
        side13 = distance point1 point3
        side23 = distance point2 point3
        side34 = distance point3 point4
        side41 = distance point4 point1

-- Функция для вычисления расстояния между двумя точками
distance :: Point -> Point -> Double
distance (x1, y1) (x2, y2) = sqrt ((x2 - x1)^2 + (y2 - y1)^2)

-- Поиск центра отрезка по координатам его концов
centre :: Point -> Point -> Point
centre (x1, y1) (x2, y2) = ((x1 + x2) / 2, (y1 + y2) / 2)

-- Вывод введённых данных
showInput :: ([Point], Circle) -> IO ()
showInput (rhombus, (circleCentre, circleRadius)) = do
    putStrLn $ "\n\27[32m> " ++ "Inputed data" ++ "\27[0m"
    putStrLn " Rhombus:"
    mapM_ (putStrLn . (\(x, y) -> "\t(" ++ show x ++ ", " ++ show y ++ ")")) rhombus
    putStrLn " Circle:"
    (putStrLn . (\(x, y) -> "\t(" ++ show x ++ ", " ++ show y ++ ")")) circleCentre
    putStrLn $ "\t" ++ show circleRadius

-- Вывод ответа
showAnswer :: [Point] -> IO ()
showAnswer [] = putStrLn $ "\n\27[31m> " ++ "Answer not found" ++ "\27[0m"
showAnswer answer = do
    putStrLn $ "\n\27[32m> " ++ "Answer" ++ "\27[0m"
    mapM_ (putStrLn . (\(x, y) -> "\t(" ++ show x ++ ", " ++ show y ++ ")")) answer

-- Функция для решения квадратного уравнения: Ax^2 + Bx + C = 0
solveQuadratic :: Double -> Double -> Double -> Maybe (Double, Double)
solveQuadratic a b c
    | discriminant < 0 = Nothing  -- Нет действительных корней
    | otherwise = Just (x1, x2)
  where
    discriminant = b * b - 4 * a * c
    sqrtDiscriminant = sqrt discriminant
    x1 = (-b + sqrtDiscriminant) / (2 * a)
    x2 = (-b - sqrtDiscriminant) / (2 * a)

-- Функция для вычисления Y по уравнению прямой: y = kx + b
lineEquation :: Double -> Double -> Double -> Double
lineEquation k b x = k * x + b
```

```haskell
-- Функция для нахождения пересечения прямой и окружности
intersection :: (Double, Double, Double) -> Circle -> Maybe (Point, Point)
intersection (k, g, b) ((centreX, centreY), radius)
    | g /= 0 = do
        let aq = (k * k) / (g * g) + 1
            bq = (2 * k * (b + centreY * g) / (g * g)) - 2 * centreX
            cq = (b / g + centreY) * (b / g + centreY) + (centreX * centreX) - radius
* radius
        (x1, x2) <- solveQuadratic aq bq cq
        let y1 = lineEquation k b x1
            y2 = lineEquation k b x2
        return ((x1, y1), (x2, y2))
    | otherwise = do
        let aq = 1
            bq = - (2 * centreY)
            cq = (b - centreX) * (b - centreX) - radius * radius + centreY * centreY
        (y1, y2) <- solveQuadratic aq bq cq
        let x1 = b
            x2 = b
        return ((x1, y1), (x2, y2))

-- Функция для проверки, лежит ли точка на отрезке
isPointOnSegment :: Point -> Point -> Point -> Bool
isPointOnSegment (x, y) (x1, y1) (x2, y2) =
    -- Проверка, что точка лежит на прямой, проходящей через (x1, y1) и (x2, y2)
    abs crossProduct < epsilon &&
    -- Проверка, что точка находится в пределах отрезка
    x >= min x1 x2 - epsilon && x <= max x1 x2 + epsilon &&
    y >= min y1 y2 - epsilon && y <= max y1 y2 + epsilon
  where
    -- Векторное произведение для проверки коллинеарности
    crossProduct = (x2 - x1) * (y - y1) - (y2 - y1) * (x - x1)
    -- Погрешность для сравнения чисел с плавающей точкой
    epsilon = 1e-10

-- Функция для проверки, лежит ли точка на сторонах ромба
isPointOnRhombus :: Point -> [Point] -> Bool
isPointOnRhombus point [p1, p2, p3, p4] =
    isPointOnSegment point p1 p2 ||
    isPointOnSegment point p2 p3 ||
    isPointOnSegment point p3 p4 ||
    isPointOnSegment point p4 p1
```