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)

Лабораторная работа №7
по курсу
«Логическое и функциональное программирование»

Выполнил:
студент группы ИКПИ-11
Дунаев В.Е.

Принял:
доцент кафедры ПИиВТ
Ерофеев С.А.

Санкт-Петербург
2023 г.

Цель работы

Разработать программу на языке Haskell, определяющую точки пересечения ромба и квадрата.

Описание алгоритма

Сначала последовательно вводим координаты фигуры. Смотрим на пересечение диагоналей и проверяем перпендикулярны они или нет. Это одно из свойств ромба. Потом измеряем длинны сторон, если они все одинаковы, то эта фигура является ромбом. Далее вводим координаты квадрата. Он является частным случаем ромба, у которого все углы равны 90 градусам. Поэтому проверяем 2 противоположных угла и подтверждаем фигуру. Прямой угол опрелеляется предикатом deg90. Если произведение векторов равно 0, то отрезки перпендикулярны.

Теперь приступаем к перебору точек пересечения. Используемый предикат isCross принимает координаты двух отрезков и проверяет пересекаются ли они. Если точки обоих отрезков лежат по обе стороны обоих отрезков, то прямые пересекаются. Далее с помощью предиката cross находим точки пересечения, используя векторы и уравнения прямых находим нужные коофициенты. Подставляя коэффициенты в уравнение, получаем точку пересечения 2 отрезков. Если прямые совпадают алгоритм выдает первую и последнюю точки пересечения.

В результате появятся много дубликатов точек, но с помощью динамической базы данных можно отсеять лишние. Ответ выводится в отдельном окошке.

Используемые функции

deg90 — проверяет есть ли 90 градусов между 2 отрезками

cross — определяет точку пересечения 2 отрезков

crossCheck — вспомогательная функция для cross

isCross — проверяет пересекаются ли 2 отрезка

point — считывает 1 точку

insertSquare — ввод точек для квадрата, с проверкой

insertRhombus — ввод точек для ромба, с проверкой

my_length — вычисляет длинну отрезка

sameLength — проверяет на одинаковость длины 2 отрезков

isRhombus — говорит является ли заданная фигура ромбом

isRhombusBool - проверяет является ли заданная фигура ромбом

isSquare — говорит является ли заданная фигура квадратом

isSquareBool - проверяет является ли заданная фигура квадратом

intersections — находит точки пересечения всех прямых заданных фигур

result — объединяет ввод координат фигур и нахождение точек их пересечения

cosoe — находит косое произведение векторов

onLine — смотрит, лежит ли точка на отрезке

onLineCheck — вспомогательная функция для onLine

inLine — проверяет, лежит ли точка в пределах отрезка

huh — находится ли координата в заданном диапазоне

onBothSides — проверяет лежат ли точки отрезков по обе стороны 2 отрезков

checkInf — проверяет 2 отрезка на бесконечное кол-во точек

printCheck — выводит найденные точки, проверяя на пустоту

printList — выводит список точек

point — вводит одну точку

main — основная функция

Тестирование

1) Квадрат (2,2 2,5 5,5 5,2)
   Ромб(2,0 2,2 4,2 4,0)

```
Бесконечное число точек
Точка = (2.0,2.0)
Точка = (4.0,2.0)
```

4) Квадрат (0,0 1,0 1,1 0,1)
   Ромб(2,0 2,2 4,2 4,0)

```
Точки пересечения не найдены
ghci>
```

2) Квадрат (2,2 2,5 5,5 5,2)
Ромб(2,5 5,8 8,5 5,2)

```
Точка = (2.0,5.0)
Точка = (5.0,2.0)
```

3) Квадрат (2,2 2,5 5,5 5,2)
Ромб(4,6 6,8 8,6 6,4)

```
Точка = (5.0,5.0)
```

5) Квадрат (0,0 1,0 1,1 0,1)
Ромб(0,0 1,0 1,1 0,1)

```
Бесконечное число точек
Бесконечное число точек
Бесконечное число точек
Точка = (0.0,0.0)
Точка = (1.0,0.0)
Точка = (1.0,1.0)
Точка = (0.0,1.0)
```

6) Квадрат (0,0 1,0 1,1 0,1)
Ромб(0,0 1,2 3,3 2,1)

```
Точка = (0.0,0.0)
Точка = (1.0,0.5)
Точка = (0.5,1.0)
```

Выводы

В ходе проведенной лабораторной работы я разработал программу на языке Haskell для определения точек пересечения ромба и квадрата. Было проведено тестирование отдельных функций и всей программы в целом. Программа полностью удовлетворяет заданным требованиям.

Исходный код программы

Seven.hs:

```
import Data.List

isSquare x1 y1 x2 y2 x3 y3 x4 y4 = do
    if (isSquareBool x1 y1 x2 y2 x3 y3 x4 y4 ) then do
        putStrLn "Вы ввели квадрат"
    else
        error "Введенная фигура не квадрат"

isRhombus x1 y1 x2 y2 x3 y3 x4 y4 = do
    if (isRhombusBool x1 y1 x2 y2 x3 y3 x4 y4 ) then do
        putStrLn "Вы ввели ромб"
    else
        error "Введенная фигура не ромб"

isSquareBool x1 y1 x2 y2 x3 y3 x4 y4 = samelen && deg90_1 && deg90_2 && deg90_3 && cross_1
    where   samelen = sameLength x1 y1 x2 y2 x3 y3 x4 y4
            deg90_1 = deg90 x1 y1 x3 y3 x2 y2 x4 y4
            deg90_2 = deg90 x1 y1 x2 y2 x2 y2 x3 y3
            deg90_3 = deg90 x1 y1 x4 y4 x4 y4 x3 y3
            cross_1 = isCross x1 y1 x3 y3 x2 y2 x4 y4

isRhombusBool x1 y1 x2 y2 x3 y3 x4 y4 = samelen && deg90_1 && cross_1
    where   samelen = sameLength x1 y1 x2 y2 x3 y3 x4 y4
            deg90_1 = deg90 x1 y1 x3 y3 x2 y2 x4 y4
            cross_1 = isCross x1 y1 x3 y3 x2 y2 x4 y4

deg90 x1 y1 x2 y2 x3 y3 x4 y4 = res == 0
    where   ab1 = x2 - x1
            ab2 = y2 - y1
            cd1 = x4 - x3
            cd2 = y4 - y3
            res = ab1*cd1 + ab2*cd2

cosoe x1 y1 x2 y2 a b = ans
    where   ab1 = x2 - x1
            ab2 = y2 - y1
            c1 = a - x1
            c2 = b - y1
            ans = ab1*c2 - c1*ab2

onBothSides x1 y1 x2 y2 a1 b1 a2 b2 = (ans1, ans2, ans3, ans4)
```

```
    where   ans1 = cosoe x1 y1 x2 y2 a1 b1
            ans2 = cosoe x1 y1 x2 y2 a2 b2
            ans3 = cosoe a1 b1 a2 b2 x1 y1
            ans4 = cosoe a1 b1 a2 b2 x2 y2

my_length x1 y1 x2 y2 = sqrt $ ((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1))

sameLength x1 y1 x2 y2 x3 y3 x4 y4 =  ans1 == ans2 && ans2 == ans3 && ans3 == ans4 && ans4 /= 0
    where   ans1 = my_length x1 y1 x2 y2
            ans2 = my_length x2 y2 x3 y3
            ans3 = my_length x3 y3 x4 y4
            ans4 = my_length x4 y4 x1 y1

inLine x1 y1 x2 y2 x y = stmt1 && stmt2
    where   stmt1 = huh x1 x2 x
            stmt2 = huh y1 y2 y

huh x1 x2 p = p >= mini && p <= maxi
    where   maxi = max x1 x2
            mini = min x2 x1

isCross x1 y1 x2 y2 a1 b1 a2 b2 = ans1*ans2 < 0 && ans3*ans4 < 0
    where   (ans1, ans2, ans3, ans4) = onBothSides x1 y1 x2 y2 a1 b1 a2 b2

checkInf x1 y1 x2 y2 x3 y3 x4 y4 =
    let ans1 = cosoe x1 y1 x2 y2 x3 y3
        inline1 = inLine x1 y1 x2 y2 x3 y3
        ans2 =  cosoe x1 y1 x2 y2 x4 y4
        inline2 = inLine x1 y1 x2 y2 x4 y4
        ans3 = cosoe x3 y3 x4 y4 x1 y1
        inline3 = inLine x3 y3 x4 y4 x1 y1
        ans4 = cosoe x3 y3 x4 y4 x2 y2
        inline4 = inLine x3 y3 x4 y4 x2 y2
    in  if ans1 == 0 && inline1 && ans2 == 0 && inline2 then
            putStrLn "Бесконечное число точек"
        else if ans3 == 0 && inline3 && ans4 == 0 && inline4 then
            putStrLn "Бесконечное число точек"
        else
            putStr ""

onLine x1 y1 x2 y2 x3 y3 x4 y4 =
    let ans1 = cosoe x1 y1 x2 y2 x3 y3
        inline1 = inLine x1 y1 x2 y2 x3 y3
        ans2 =  cosoe x1 y1 x2 y2 x4 y4
        inline2 = inLine x1 y1 x2 y2 x4 y4
        ans3 = cosoe x3 y3 x4 y4 x1 y1
        inline3 = inLine x3 y3 x4 y4 x1 y1
    in  if ans1 == 0 && inline1 then
            (x3, y3)
        else if ans2 == 0 && inline2 then
            (x4, y4)
        else if ans3 == 0 && inline3 then
            (x1, y1)
        else
            (x2, y2)

onLineCheck x1 y1 x2 y2 x3 y3 x4 y4 =
    let ans1 = cosoe x1 y1 x2 y2 x3 y3
        inline1 = inLine x1 y1 x2 y2 x3 y3
        ans2 =  cosoe x1 y1 x2 y2 x4 y4
        inline2 = inLine x1 y1 x2 y2 x4 y4
        ans3 = cosoe x3 y3 x4 y4 x1 y1
        inline3 = inLine x3 y3 x4 y4 x1 y1
        ans4 = cosoe x3 y3 x4 y4 x2 y2
        inline4 = inLine x3 y3 x4 y4 x2 y2
    in  if ans1 == 0 && inline1 then
            True
        else if ans2 == 0 && inline2 then
            True
        else if ans3 == 0 && inline3 then
            True
        else if ans4 == 0 && inline4 then
            True
        else
            False

crossCheck x1 y1 x2 y2 x3 y3 x4 y4 =
    let my_flag = isCross x1 y1 x2 y2 x3 y3 x4 y4
        temp1 = y2 - y1
        temp2 = y3 - y4
        q = (x2 - x1) / (y1 - y2)
        sn = (x3 - x4) + (y3 - y4) * q
    in  if my_flag && temp1 /= 0 && sn /= 0 then
            True
        else if my_flag && temp1 == 0 && temp2 /= 0 then
            True
        else
            onLineCheck x1 y1 x2 y2 x3 y3 x4 y4

cross x1 y1 x2 y2 x3 y3 x4 y4 =
    let temp1 = y2 - y1
        temp2 = y3 - y4
        q = (x2 - x1) / (y1 - y2)
        sn = (x3 - x4) + (y3 - y4) * q
        fn = (x3 - x1) + (y3 - y1) * q
        n1 = fn / sn
        n2 = (y3 - y1) / (y3 - y4)
    in  if temp1 /= 0 && sn /= 0 then
            ((x3 + (x4 - x3) * n1), (y3 + (y4 - y3) * n1))
        else if temp1 == 0 && temp2 /= 0 then
            ((x3 + (x4 - x3) * n2), (y3 + (y4 - y3) * n2))
        else
            onLine x1 y1 x2 y2 x3 y3 x4 y4

intersections x1 y1 x2 y2 x3 y3 x4 y4 a1 b1 a2 b2 a3 b3 a4 b4 = do
    let part1 = if crossCheck x1 y1 x2 y2 a1 b1 a2 b2 then do
                    let (a,b) = cross x1 y1 x2 y2 a1 b1 a2 b2 in
                        [(a,b)]
                else []
    checkInf x1 y1 x2 y2 x3 y3 x4 y4
    let part2 = part1 ++ if crossCheck x1 y1 x2 y2 a2 b2 a3 b3 then
                    let (a,b) = cross x1 y1 x2 y2 a2 b2 a3 b3 in
                        [(a,b)]
                else []
    checkInf x1 y1 x2 y2 a2 b2 a3 b3
    let part3 = part2 ++ if crossCheck x1 y1 x2 y2 a3 b3 a4 b4 then
                    let (a,b) = cross x1 y1 x2 y2 a3 b3 a4 b4 in
                        [(a,b)]
                else []
    checkInf x1 y1 x2 y2 a3 b3 a4 b4
    let part4 = part3 ++ if crossCheck x1 y1 x2 y2 a4 b4 a1 b1 then
                    let (a,b) = cross x1 y1 x2 y2 a4 b4 a1 b1 in
                        [(a,b)]
                else []
    checkInf x1 y1 x2 y2 a4 b4 a1 b1
```

```
    let part5 = part4 ++ if crossCheck x2 y2 x3 y3 a1 b1 a2 b2 then
                   let (a,b) = cross x2 y2 x3 y3 a1 b1 a2 b2 in
                        [(a,b)]
                else []
    checkInf x2 y2 x3 y3 a1 b1 a2 b2
    let part6 = part5 ++ if crossCheck x2 y2 x3 y3  a2 b2 a3 b3 then
                   let (a,b) = cross x2 y2 x3 y3  a2 b2 a3 b3 in
                        [(a,b)]
                else []
    checkInf x2 y2 x3 y3  a2 b2 a3 b3
    let part7 = part6 ++ if crossCheck x2 y2 x3 y3 a3 b3 a4 b4 then
                   let (a,b) = cross x2 y2 x3 y3 a3 b3 a4 b4 in
                        [(a,b)]
                else []
    checkInf x2 y2 x3 y3 a3 b3 a4 b4
    let part8 = part7 ++ if crossCheck x2 y2 x3 y3 a4 b4 a1 b1 then
                   let (a,b) = cross x2 y2 x3 y3 a4 b4 a1 b1 in
                        [(a,b)]
                else []
    checkInf x2 y2 x3 y3 a4 b4 a1 b1

    let part9 = part8 ++ if crossCheck x3 y3 x4 y4 a1 b1 a2 b2 then
                   let (a,b) = cross x3 y3 x4 y4 a1 b1 a2 b2 in
                        [(a,b)]
                else []
    checkInf x3 y3 x4 y4 a1 b1 a2 b2
    let part10 = part9 ++ if crossCheck x3 y3 x4 y4 a2 b2 a3 b3 then
                   let (a,b) = cross x3 y3 x4 y4 a2 b2 a3 b3 in
                        [(a,b)]
                else []
    checkInf x3 y3 x4 y4 a2 b2 a3 b3
    let part11 = part10 ++ if crossCheck x3 y3 x4 y4 a3 b3 a4 b4 then
                   let (a,b) = cross x3 y3 x4 y4 a3 b3 a4 b4 in
                        [(a,b)]
                else []
    checkInf x3 y3 x4 y4 a3 b3 a4 b4
    let part12 = part11 ++ if crossCheck x3 y3 x4 y4 a4 b4 a1 b1 then
                   let (a,b) = cross x3 y3 x4 y4 a4 b4 a1 b1 in
                        [(a,b)]
                else []
    checkInf x3 y3 x4 y4 a4 b4 a1 b1

    let part13 = part12 ++ if crossCheck x4 y4 x1 y1 a1 b1 a2 b2 then
                   let (a,b) = cross x4 y4 x1 y1 a1 b1 a2 b2 in
                        [(a,b)]
                else []
    checkInf x4 y4 x1 y1 a1 b1 a2 b2
    let part14 = part13 ++ if crossCheck x4 y4 x1 y1 a2 b2 a3 b3 then
                   let (a,b) = cross x4 y4 x1 y1 a2 b2 a3 b3 in
                        [(a,b)]
                else []
    checkInf x4 y4 x1 y1 a2 b2 a3 b3
    let part15 = part14 ++ if crossCheck x4 y4 x1 y1 a3 b3 a4 b4 then
                   let (a,b) = cross x4 y4 x1 y1 a3 b3 a4 b4 in
                        [(a,b)]
                else []
    checkInf x4 y4 x1 y1 a3 b3 a4 b4
    let part16 = part15 ++ if crossCheck x4 y4 x1 y1 a4 b4 a1 b1 then
                   let (a,b) = cross x4 y4 x1 y1 a4 b4 a1 b1 in
                        [(a,b)]
                else []
    checkInf x4 y4 x1 y1 a4 b4 a1 b1
    printCheck $ nub part16


printCheck [] = putStrLn "Точки пересечения не найдены"
printCheck lst = printList lst

printList [] = putStrLn ""
printList (x:xs) = do
    putStr "Точка = "
    putStrLn $ show x
    printList xs

point = do
    putStr "X = "
    x <- readLn :: IO Double
    putStr "Y = ":
    y <- readLn :: IO Double
    return (x, y)


insertSquare = do
    putStrLn "Введите квадрат"
    putStrLn "Точка 1"
    (x1,y1) <- point
    putStrLn "Точка 2"
    (x2,y2) <- point
    putStrLn "Точка 3"
    (x3,y3) <- point
    putStrLn "Точка 4"
    (x4,y4) <- point
    isSquare x1 y1 x2 y2 x3 y3 x4 y4
    return (x1, y1, x2, y2, x3, y3, x4, y4)


insertRhombus = do
    putStrLn "Введите ромб"
    putStrLn "Точка 1"
    (x1,y1) <- point
    putStrLn "Точка 2"
    (x2,y2) <- point
    putStrLn "Точка 3"
    (x3,y3) <- point
    putStrLn "Точка 4"
    (x4,y4) <- point
    isRhombus x1 y1 x2 y2 x3 y3 x4 y4
    return (x1, y1, x2, y2, x3, y3, x4, y4)


main :: IO()
main = do
    (x1, y1, x2, y2, x3, y3, x4, y4) <- insertSquare
    (a1, b1, a2, b2, a3, b3, a4, b4) <- insertRhombus
    intersections x1 y1 x2 y2 x3 y3 x4 y4 a1 b1 a2 b2 a3 b3 a4 b4
```