**Условие задачи**

Исследователь в центре реабилитации изучает использование пациентом инвалидной коляски с мотором на ограниченной площадке около здания центра. Мотор через привод соединен с осью, поэтому оба колеса вращаются с одинаковой скоростью, и коляска может ехать только по прямой. Пациент может останавливать коляску и поворачивать колеса, и, таким образом, он может изменить направление движения только когда коляска остановлена. Для более удобного наблюдения за коляской она оборудована компасом, часами и спидометром. Записывающее устройство записывает каждый временной промежуток, в течение которого двигалась коляска, среднюю скорость движения и показания компаса во время этого промежутка. Компас стандартный, т.е. 0° соответствует северу, 90° – востоку, и тд.

Карта ограниченной площадки показана на рисунке. Площадка представляет собой прямоугольную часть лужайки размером 200 на 400 футов. Пациент въезжает на площадку через ворота здания, расположенного на южной границе участка. Ворота расположены посередине южной границы, как показано на рисунке.

Записывающее устройство включается при въезде на площадку и отображает передвижения пациента в течение одного часа. Время измеряется в секундах, начиная с нуля, что соответствует моменту прохождения коляской ворот. Для описания движения коляски в течение каждого промежутка, когда работает мотор, устройство записывает четыре числа. Первый два обозначают время начала и конца передвижения, третье – среднюю скорость в течение этого временного промежутка, и четвертое – показание компаса (в течение каждого интервала коляска сохраняет постоянные скорость и направление движения). Например, запись

10.6 15.9 2.0 274

обозначает, что в течение времени от t1 = 10.6 до t2 = 15.9 секунд коляска двигалась со скоростью 2.8 футов в секунду в направлении 274°. Время записывается с точностью 0.1 секунды, скорость – 0.1 фута в секунду, а показания компаса – целым числом градусов.

Ваша задача – проанализировать данные, записанные устройством. В частности, нужно определить следующее:

1. Покидал ли пациент ограниченную площадку? Если да, то необходимо определить время, когда он выехал за пределы площадки, и точку на периметре участка, где была пересечена граница. Если пациент не покидал площадку, то на какое максимальное расстояние он удалялся от ворот?
2. Какова длина пути, проделанного пациентом?

Для ответа на вопросы введите систему координат с началом в юго-западном углу площадки и координатами северо-восточного угла (400,200). Так как устройство включается при прохождении ворот, положение пациента в момент времени 0.0 всегда будет задаваться координатами (200,0). После въезда на площадку пациент сначала будет двигаться на север.

**Ввод**

Вводимая информация состоит из нескольких блоков. В первой строке каждого блока данных записано целое число, обозначающее количество строк в этом блоке. Каждая следующая строка состоит из четырех чисел, записанных устройством в течение текущего промежутка времени. Конец ввода данных обозначается блоком, первая строка которого содержит ноль.

В первом блоке данных примера ввода пациент проехал через ворота (в момент времени 0.0) и первые пять секунд двигался на север со скоростью 3 фута в секунду. Затем с времени t = 7 до t = 9 он ехал со скоростью 2 фута в секунду в направлении 30°. Потом он остановился, изменил направление на 60° и двигался со скоростью 4 фута в секунду с t = 10 до t = 100 секунд. Десять секунд спустя (в t = 110) он поехал на север со скоростью 2 фута в секунду до момента времени t = 200.

**Вывод**

Вывод для каждого блока данных начинается с номера. Должно быть указано, покидал ли пациент площадку, если да, то в какой момент времени и в какой точке периметра он пересек границу. Если же нет, то должна быть указана максимальная дистанция, на которую он удалился от ворот. В любом случае также нужно вывести длину пути. Проделанного пациентом. Выводите данные в формате, показанном в примере вывода, разделяя различные опыты строкой звездочек.

**Пример ввода**

4

|  |  |  |  |
| --- | --- | --- | --- |
| 0.0 | 5.0 | 3.0 | 0 |
| 7.0 | 9.0 | 2.0 | 30 |
| 10.0 | 100.0 | 4.0 | 60 |
| 110.0 | 200.0 | 2.0 | 0 |

3

|  |  |  |  |
| --- | --- | --- | --- |
| 0.0 | 20.0 | 2.0 | 0 |
| 500.0 | 600.0 | 1.0 | 270 |
| 3000.0 | 3100.0 | 1.0 | 0 |

7

|  |  |  |  |
| --- | --- | --- | --- |
| 0.0 | 5.3 | 2.1 | 0 |
| 19.0 | 35.6 | 2.7 | 346 |
| 42.0 | 70.4 | 2.3 | 15 |
| 1101.4 | 1192.1 | 1.7 | 117 |
| 2107.0 | 2193.6 | 2.1 | 295 |
| 2196.3 | 2201.2 | 2.0 | 290 |
| 2704.3 | 2709.2 | 1.5 | 208 |

0

**Пример вывода**

Случай номер 1.  
Коляска покинула ограниченную площадку в точке (400.0, 132.8) спустя 67.2 секунды после старта.  
Общий пройденный путь равен 559.0 фута.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Случай номер 2.  
Коляска не покидала ограниченную площадку.  
Максимальное удаление коляски от точки старта 172.0 фута.  
Общий пройденный путь равен 240.0 фута.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Случай номер 3  
Коляска покинула ограниченную площадку в точке (67.0,200.0) спустя 2191.4 секунды после старта.  
Общий пройденный путь равен 354.7 фута.

**Схема программы**

нет

Конец

event\_number += 1  
x, y = 200, 0  
point\_of\_intersection = [100, 100]  
time\_of\_intersection = -1  
max\_distance = 0  
all\_distance = 0

Выводим данные о случае в формате, описанном в условии

i = 0

да

нет

Вызов функции moving(\*data[i])

да

type(data[i]) is int

i от 0 до длины data - 1

Считываем данные из файла data.txt и формируем список, преобразуя отдельные числа в int, а списки чисел в list.

point\_of\_intersection = [100, 100]  
time\_of\_intersection = -1  
event\_number = 0  
max\_distance = 0  
all\_distance = 0

Начало

**Функция moving(t1, t2, speed, direction)**

Начало

Конец

all\_distance += sqrt((sin(direction) \* distance)\*\*2 + (cos(direction) \* distance)\*\*2)

нет

да

нет

нет

max\_distance = distnt

distnt > max\_distance

distn = sqrt((x – 200)\*\*2 + y\*\*2)

point\_of\_intersection = intersection[:2]  
time\_of\_intersection = intersection[-1]

да

Intersection is True

intersection = False  
вызов функции check\_intersection(x – sin(direction) \* distance, y – cos(direction) \* distance, x, y, t1, speed)

distance = (t2 – t1) \* speed  
x += sin(direction) \* distance  
y += cos(direction) \* distance

да

time\_of\_intersection == -1

**Функция check\_intersection(x1, y1, x2, y2, t1, speed)**

Начало

displacement\_vector\_x = x2 – x1  
displacement\_vector\_y = y2 – y1

displacement\_vector\_x != 0

нет

да

y = displacement\_vector\_y \* (-x1 / displacement\_vector\_x) + y1

да

A

displacement\_vector\_y != 0

нет

distance\_to\_intersection = sqrt((400 – x1)\*\*2 + (y – y1)\*\*2)  
intersection = [400, y, t1 + (distance\_to\_intersection / speed)]

да

0 <= y <= 200 and x1 <= 400 <= x2 and min(y2, y1) <= y <= max(y2, y1)

нет

y = displacement\_vector\_y \* ((400 – x1) / displacement\_vector\_x) + y1

distance\_to\_intersection = sqrt(x1\*\*2 + (y – y1)\*\*2)  
intersection = [0, y, t1 + (distance\_to\_intersection / speed)]

да

0 <= y <= 200 and x2 <= 0 <= x1 and min(y2, y1) <= y <= max(y2, y1)

нет

B

Конец

B

distance\_to\_intersection = sqrt((x – x1)\*\*2 + x\*\*2)  
intersection = [x, 200, t1 + (distance\_to\_intersection / speed)]

да

0 <= x <= 400 and y1 <= 200 <= y2 and min(x1, x2) <= x <= max(x1, x2)

x = displacement\_vector\_x \* ((200 – y1) / displacement\_vector\_y) + x1

нет

distance\_to\_intersection = sqrt((x – x1)\*\*2 + y1\*\*2)  
intersection = [x, 0, t1 + (distance\_to\_intersection / speed)]

да

0 <= x <= 400 and y2 <= 0 <= y1 and min(x1, x2) <= x <= max(x1, x2)

x = displacement\_vector\_x \* (-y1 / displacement\_vector\_y) + x1

А

**Описание программы**

Считав из файла необходимые данные, мы помещаем их в список data. Например,

data = [4, [0.0, 5.0, 3.0, 0.0], [7.0, 9.0, 2.0, 30.0], [10.0, 100.0, 4.0, 60.0], [110.0, 200.0, 2.0, 0], 0]

Далее мы проходим в цикле по всему списку, и если нам встречается отдельное число – значит, что далее пойдут данные о перемещениях коляски. И мы должны, во-первых, напечатать данные о предыдущем случае (если он был), а во-вторых - обнулить все необходимые переменные (опять же если у нас до этого они были.

У нас существуют 6 глобальных переменных:

* x, y – координаты коляски
* point\_of\_intersection – координаты пересечения с линиями площадки
* time\_of\_intersection – время пересечения площадки после старта
* max\_distance – максимальное расстояние между коляской и воротами
* all\_distance – общее пройденное расстояние

Использование глобальных переменных помогает более удобным способом реализовать взаимодействие между функциями, т.к. для каждого отдельного действия нам не нужно передавать в функции все данные – есть общие глобальные переменные. В больших проектах такого делать не рекомендуется, но для моего это неплохое решение.

Далее про логику функций:

* **moving(t1, t2, speed, direction)** – основная функция, которая отвечает за перемещение коляски и вызывает все сопутствующие функции для отслеживания пересечения (check\_intersection), подсчета максимального расстояния (get\_max\_distance) и др. Функция ничего не возвращает, т.к. работает с глобальными переменными.
* **check\_intersection(x1, y1, x2, y2, t1, speed)** – функция проверяет, пересекла ли коляска ограниченную площадку при перемещении из точки (x1, y1) в точку (x2, y2). Переменные t1 и speed необходимы, чтобы в случае чего подсчитать время пересечения коляски с площадкой. В случае пересечения возвращает список, состоящий из точек пересечения с площадкой и временем пересечения. В случае если коляска не пересекает площадку, возвращает False
* **get\_max\_distance(x2, y2)** – функция просто ищет максимальное расстояние между коляской и воротами. Работает с глобальной переменной max\_distance
* **get\_vector\_length(x1, y1, x2, y2)** – создана для удобства подсчета длины вектора. Возвращает его длину.
* **сheck\_input\_data(data)** – проверка корректности введенных данных.

Рассмотрим подробнее алгоритм поиска пересечений коляски с площадкой, т.к. в данной программе он оказался самым сложно реализуемым среди остальных довольно простых механик.

Т.к. каждую стенку площадки и каждый вектор перемещения мы можем задать с помощью уравнения прямой, поиск пересечения сводится к нахождению точки пересечения прямой, содержащей вектор перемещения и прямой, соответствующей какой-либо из стенок. Рассмотрим на примере. Допустим, у нас коляска перемещается из точки (50, 50) в точку (-90, 190). Составим каноническое уравнение данной прямой, взяв за направляющий вектор s = {-90 – 50; 190 - 50}, а за M₀ точку начального положения коляски (50, 50). Тогда получим уравнение:

(x - 50) / (-140) = (y - 50) / (140) (\*)

Проверим пересечение допустим с левой стенкой. Уравнение прямой, содержащей левую стенку, выглядит так: x = 0

Подставив в уравнение (\*) x = 0 получим следующее:

y = (-50 \* 140) / (-140) + 50 = 100

Далее нам необходимо проверить, принадлежит ли полученная точка пересечения ограниченному отрезку нашей площадки, т.к. уравнения прямых – бесконечны и может случиться такое, что вектор перемещения, образующий прямую, пересекает левую стенку далеко вверху, за пределами площадки.

Думаю, основная идея понятна. Далее не составляет труда составить алгоритм проверки пересечения коляски с каждой из 4 стен.

**Текст программы**

**Файл main.py**

from math import cos, sin, sqrt, radians  
from global\_variables import \*  
  
  
def get\_vector\_length(x1, y1, x2, y2): # получение длины вектора  
 return sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2)  
  
  
def check\_input\_data(data): # проверка правильности введенных данных  
 data = data.split('\n')  
 try:  
 int(data[0])  
 except ValueError:  
 return False  
 k = 0  
 while k < len(data) - 1:  
 try:  
 # считаем, правильное ли количество шагов указано  
 input\_count\_steps, find\_count\_steps = int(data[k]), 0  
 for i in range(int(data[k])):  
 try:  
 now\_step = list(map(float, data[k + i + 1].split()))  
 # если в описывающем перемещение случае не 4 параметра  
 if len(now\_step) != 4:  
 return False  
 if i == 0 and now\_step[0] != 0: # если начальное время не равно 0 секундам  
 return False  
 find\_count\_steps += 1  
 except ValueError:  
 return False  
 if input\_count\_steps != find\_count\_steps:  
 return False  
 k += 1 + find\_count\_steps  
 except ValueError: # если какое-либо из преобразований вызывало ошибку,  
 # значит, введенные данные некорректны  
 return False  
 try:  
 if k != len(data) - 1 or int(data[k]) != 0: # последним значением должен быть ноль  
 return False  
 except ValueError:  
 return False  
 return True  
  
  
# функция проверки пересечения с границами площадки.  
# x1, y1 - координаты до перемещения, x2, y2 - после  
def check\_intersection(x1, y1, x2, y2, t1, speed):  
 displacement\_vector\_x, displacement\_vector\_y = x2 - x1, y2 - y1  
  
 if displacement\_vector\_x != 0:  
 # проверка пересечения с левой стенкой  
 y = displacement\_vector\_y \* (-x1 / displacement\_vector\_x) + y1  
 if 0 <= y <= 200 and x2 <= 0 <= x1 and min(y2, y1) <= y <= max(y2, y1):  
 distance\_to\_intersection = get\_vector\_length(x1, y1, 0, y)  
 # возвращаем координату пересечения с левой стенкой и время пересечения  
 return [0, y, t1 + (distance\_to\_intersection / speed)]  
  
 # проверка пересечения с правой стенкой  
 y = displacement\_vector\_y \* ((400 - x1) / displacement\_vector\_x) + y1  
 if 0 <= y <= 200 and x1 <= 400 <= x2 and min(y2, y1) <= y <= max(y2, y1):  
 distance\_to\_intersection = get\_vector\_length(x1, y1, 400, y)  
 # возвращаем координату пересечения с правой стенкой и время пересечения  
 return [400, y, t1 + (distance\_to\_intersection / speed)]  
  
 if displacement\_vector\_y != 0:  
 # проверка пересечения с нижней стенкой  
 x = displacement\_vector\_x \* (-y1 / displacement\_vector\_y) + x1  
 if 0 <= x <= 400 and y2 <= 0 <= y1 and min(x1, x2) <= x <= max(x1, x2):  
 distance\_to\_intersection = get\_vector\_length(x1, y1, x, 0)  
 # возвращаем координату пересечения с нижней стенкой и время пересечения  
 return [x, 0, t1 + (distance\_to\_intersection / speed)]  
  
 # проверка пересечения с верхней стенкой  
 x = displacement\_vector\_x \* ((200 - y1) / displacement\_vector\_y) + x1  
 if 0 <= x <= 400 and y1 <= 200 <= y2 and min(x1, x2) <= x <= max(x1, x2):  
 distance\_to\_intersection = get\_vector\_length(x1, y1, x, 200)  
 # возвращаем координату пересечения с верхней стенкой  
 return [x, 200, t1 + (distance\_to\_intersection / speed)]  
 return False  
  
  
def get\_max\_distance(x2, y2):  
 global max\_distance  
 distance = get\_vector\_length(200, 0, x2, y2)  
 if distance > max\_distance:  
 max\_distance = distance  
  
  
def moving(t1, t2, speed, direction): # функция перемещения коляски  
 global x, y, point\_of\_intersection, time\_of\_intersection, all\_distance  
 distance = (t2 - t1) \* speed  
 x += sin(radians(direction)) \* distance  
 y += cos(radians(direction)) \* distance  
 if time\_of\_intersection == -1:  
 intersection = check\_intersection(x - sin(radians(direction)) \* distance,  
 y - cos(radians(direction)) \* distance, x, y, t1,  
 speed)  
 if intersection:  
 point\_of\_intersection = intersection[:2]  
 time\_of\_intersection = intersection[-1]  
 else:  
 get\_max\_distance(x, y) # если нет пересечения, проверяем расстояние от ворот  
 all\_distance += get\_vector\_length(x - sin(radians(direction)) \* distance,  
 y - cos(radians(direction)) \* distance, x, y)  
  
  
def main():  
 global x, y, point\_of\_intersection, time\_of\_intersection, max\_distance, all\_distance  
 data = open(input\_file\_name, 'r',  
 encoding='utf-8').read() # считываем из файла входные данные  
 if check\_input\_data(  
 data): # если введенные данные корректны, то продолжаем работу программы  
 data = data.split('\n')  
 for i in range(len(data)):  
 if len(data[i].split()) == 1:  
 data[i] = int(data[i])  
 else:  
 data[i] = list(map(float, data[i].split()))  
  
 event\_number = 0  
 for i in range(len(data)):  
 if type(data[i]) is int:  
 if i != 0:  
 print('Случай номер {}.'.format(event\_number))  
 if time\_of\_intersection == -1:  
 print('Коляска не покидала ограниченную площадку.')  
 else:  
 print(  
 'Коляска покинула ограниченную площадку в точке ({:.2f}, '  
 '{:.2f}) спустя {:.1f} '  
 'секунды после старта.'.format(point\_of\_intersection[0],  
 point\_of\_intersection[1],  
 time\_of\_intersection))  
 print('Общий пройденный путь равен {:.1f} фута.'.format(all\_distance))  
 print('\*' \* 100)  
 event\_number += 1  
 x, y = 200, 0  
 point\_of\_intersection = [100, 100]  
 time\_of\_intersection = -1  
 max\_distance = 0  
 all\_distance = 0  
 else:  
 moving(\*data[i])  
 else:  
 print('Введенные вами данные некорректны.')

if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Файл global\_variables.py**

x, y = 200, 0 # координаты коляски  
point\_of\_intersection = [100, 100] # координаты пересечения с линиями площадки  
time\_of\_intersection = -1 # время пересечения площадки после старта  
max\_distance = 0 # максимальное расстояние удаления от ворот  
all\_distance = 0 # общее пройденное расстояние  
input\_file\_name = 'data.txt'

**Заключение**

После работы с данной программы я научился: читать данные из файла, проверять эти данные на корректность. Также я попрактиковался в разработке логики относительно объемной программы, разделяя задачу на более маленькие подзадачи. И немаловажным плюсом было применение знаний аналитической геометрии для решение поставленной задачи. Я думаю, полученные навыки помогут мне в будущем обучении

**Список литературы**

1. Лекции по курсу Борисов С. В.
2. «Аналитическая геометрия» А.Н. Канатников, А.П. Крищенко, издательство МГТУ имени Н.Э. Баумана