TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu

Lab No.: 3

A Lab Report on *Polygon Convex Test, Point Inclusion Test, Ray Casting*

Submitted by:

Name: Brihat Ratna Bajracharya

Roll No.: 19/075

Submitted to:

Mr. Jagdish Bhatta

Central Department of Computer Science and Information Technology

Date of submission: ……………………

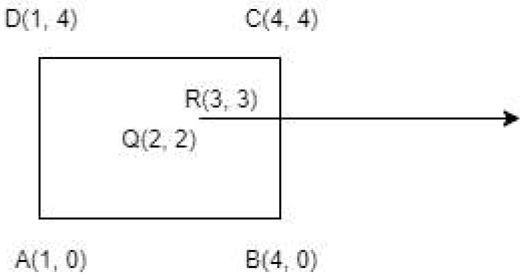
LAB 3

1. Construct a Polygon and determine whether it is convex or not
2. Perform Point Inclusion by turn test if it is convex
3. Implement Ray Casting

Code *(*`https://github.com/Brihat9/CG/blob/master/cg_lab_3.py`*)*

```
#!/usr/bin/env python

from basics import Point, LineSegment
from circular_doubly_linked_list import CircularDoublyLinkedList
from cg_lab_2_lr_turn import compute_area, is_colinear, is_left_turn
from cg_lab_2_1_line_segment_intersection import does_lines_intersect

import copy
import math

''' change input file here '''
INPUT_FILE = 'cg_lab_3_input_file_5'

def is_polygon_convex(polygon):
    """ checks whether given polygon is convex or not
        parameters: Polygon
        output: boolean
    """
    vertex_num = polygon.get_count()
    is_left_list = [False] * vertex_num

    cursor = polygon.head
    for index in range(vertex_num):
        is_left_list[index] = is_left_turn(cursor.data, cursor.next.data,
cursor.next.next.data)
        cursor = cursor.next
    # print(is_left_list)

    return True if set(is_left_list) == {True} else False

def is_point_inclusion(polygon, query_point):
    """ checks whether given point is inside given polygon or not
        parameters: Polygon, query point
        output: boolean
    """
    vertex_num = polygon.get_count()
    is_qpoint_left_turn_list = [False] * vertex_num

    cursor = polygon.head
    for index in range(vertex_num):
        # print(index, (index+1)%vertex_num, "query_point")
        is_qpoint_left_turn_list[index] = is_left_turn(cursor.data, cursor.next.data,
query_point)
        cursor = cursor.next
    # print(is_qpoint_left_turn_list)

    return True if set(is_qpoint_left_turn_list) == {True} else False

def is_vertex_colinear(ray_line, vertex):
    """ checks whether polygon vertex is colinear with ray
        using area of triangle == 0 condition
    """
    area = compute_area(ray_line.start, ray_line.terminal, vertex)
    return True if area == 0.0 else False

def check_ray_on_polygon_boundary(polygon, ray_start):
    """ checks whether ray_start lies on the boundary of polygon
        if ray_start lies on boundary, return that edge
        else return False
    """
    vertex_num = polygon.get_count()
    cursor = polygon.head
    for index in range(vertex_num):
        if is_colinear(cursor.data, cursor.next.data, ray_start):
            return LineSegment(cursor.data, cursor.next.data)
    return False

def ray_casting(polygon, ray_line):
    """ checks number of intersection a ray makes with polygon
        parameters: Polygon, ray (line)
        output: number of intersection
    """
    vertex_num = polygon.get_count()
    ray_casting_result = [False] * vertex_num

    ''' count for vertices that is colinear and intersects with ray '''
    vertex_colinear_intersect_with_ray = 0

    cursor = polygon.head
    for index in range(vertex_num):
        edge = LineSegment(cursor.data, cursor.next.data)
        ray_casting_result[index] = does_lines_intersect(edge, ray_line)
        cursor = cursor.next

        ''' added to check whether vertex is colinear with ray '''
        if is_vertex_colinear(ray_line, cursor.data) and ray_casting_result[index]:
            vertex_colinear_intersect_with_ray = vertex_colinear_intersect_with_ray + 1
    # print(ray_casting_result)
    # print(vertex_colinear_intersect_with_ray)

    ''' adjusted for colinear vertices '''
    return ray_casting_result.count(True) - vertex_colinear_intersect_with_ray

def main():
    """ Main Function """

    print("CG LAB 3")
    print("Brihat Ratna Bajracharya\n19/075\n")

    try:
        ''' reads input file '''
        in_file = open(INPUT_FILE, 'r')

        ''' get number of vertices for polygon '''
        print("Enter number of vertex of polygon:"),
        vertex_num = int(in_file.readline())
        print(vertex_num)

        ''' reads coords of point '''
        input_coords = in_file.readline()
        input_coords_list = input_coords.split()
        # print(input_coords_list)

        ''' initialize vertex list '''
        points = [None] * vertex_num

        ''' get coordinates of each vertices '''
        for index in range(vertex_num):
            print(" Enter coordinates of vertex V{}:".format(index+1)),
            input_coords_point = input_coords_list[index].split(',')
            points[index] = Point(int(input_coords_point[0]), int(input_coords_point[1]))
            print(points[index])

        ''' calculate centroid of polygon '''
        centroid = Point(sum([point.x for point in points])/len(points),sum([point.y for
point in points])/len(points))
        # print("Centroid of all Points: " + str(centroid))

        ''' sort vertices of polygon in anti-clockwise order '''
        sorted_p = copy.deepcopy(points)
        sorted_p.sort(key=lambda p: math.atan2(p.y-centroid.y,p.x-centroid.x))

        ''' show points in sorted order '''
        # print("\nPoints in sorted order")
        # for index in range(vertex_num):
        #     print(sorted_p[index]),
        # print("\n")

        polygon = CircularDoublyLinkedList()
        for index in range(vertex_num):
            polygon.append(sorted_p[index])

        ''' displays content of polygon and count of vertices '''
        polygon.display("Vertices of Polygon (sorted)")
        # polygon.show_count()


        ''' polygon convex test '''
        convex_check = is_polygon_convex(polygon)
        print("\nRESULT: Polygon is {}convex.".format('' if convex_check else 'not '))


        if convex_check:
            ''' check point inclusion only if polygon is convex '''
            print("\n\nPOINT INCLUSION BY TURN TEST")
            print("\n Enter coordinates of Query Point (P):"),

            ''' read query point from file '''
            qp = in_file.readline().split()[0].split(',')
            # print(qp)

            query_point =  Point(int(qp[0]), int(qp[1]))
            print(query_point)

            ''' point inclusion using turn test '''
            is_point_in_polygon = is_point_inclusion(polygon, query_point)
            print("\nRESULT: Query Point {}inside given polygon.".format('' if
is_point_in_polygon else 'not '))

        print("\n\nRAY CASTING")
        print("\n Enter coordinates of ray point (R):"),

        ''' read ray point start from file '''
        if not convex_check:
            in_file.readline()
        rp = in_file.readline().split()[0].split(',')
        # print(rp)
        ray_point =  Point(int(rp[0]), int(rp[1]))
        print(ray_point)

        ray_on_edge = check_ray_on_polygon_boundary(polygon, ray_point)
        if ray_on_edge:
            print("\nRESULT: Ray origin on boundary of polygon.")
            print("\tRay origin on edge joining points {}.".format(ray_on_edge))
        else:
            ''' assuming ray infinity point to the right side of polygon '''
            ray_xcoord_infinity = max([point.x for point in sorted_p])
            ray_ycoord_infinity = sum([point.y for point in sorted_p]) / vertex_num
            ray_point_infinity = Point(ray_xcoord_infinity * 100, ray_ycoord_infinity)
            ray_line_infinity = LineSegment(ray_point,ray_point_infinity)
            # print("\n  Ray Point Infinity: " + str(ray_point_infinity))
            # print("  Ray Line: " + str(ray_line_infinity))

            ''' ray intersection using line_segment_intersection test '''
            ray_intersection_count = ray_casting(polygon, ray_line_infinity)
            ray_casting_result = ray_intersection_count % 2

            print("\nRESULT: Ray origin {} of polygon.".format('inside' if ray_casting_result
else 'outside'))

        in_file.close()
        print("\nDONE.")

    except Exception as e:
        print("\n\nERROR OCCURED !!!\n Type of Error:  " + type(e).__name__)
        print(" Error message:  " + str(e.message))

if __name__ == '__main__':
    main()
```

Output 1:



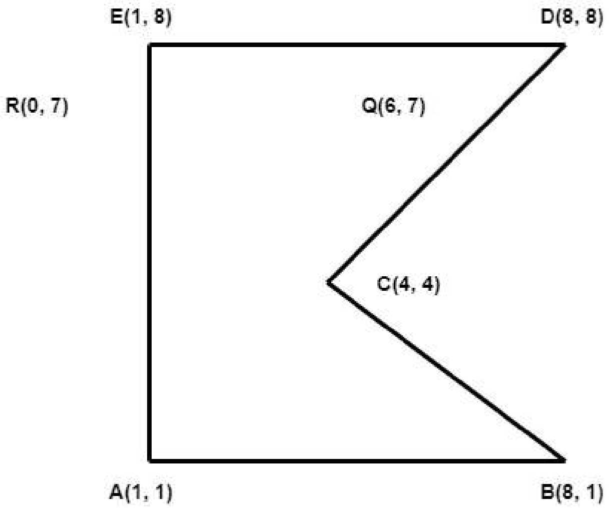
```
$ ./cg_lab_3.py

CG LAB 3
Brihat Ratna Bajracharya
19/075

Enter number of vertex of polygon: 4
 Enter coordinates of vertex V1: (1, 0)
 Enter coordinates of vertex V2: (4, 4)
 Enter coordinates of vertex V3: (1, 4)
 Enter coordinates of vertex V4: (4, 0)

Vertices of Polygon (sorted): [ (1, 0) (4, 0) (4, 4) (1, 4) ] #

RESULT: Polygon is convex.


POINT INCLUSION BY TURN TEST

 Enter coordinates of Query Point (P): (2, 2)

RESULT: Query Point inside given polygon.
```

```
RAY CASTING

 Enter coordinates of ray point (R): (3, 3)

RESULT: Ray origin inside of polygon.

DONE.
```

Input File for Output 1:

```
4
1,0 4,4 1,4 4,0
2,2
3,3
```

Output 2:



```
$ ./cg_lab_3.py

CG LAB 3
Brihat Ratna Bajracharya
19/075

Enter number of vertex of polygon: 5
 Enter coordinates of vertex V1: (1, 1)
 Enter coordinates of vertex V2: (4, 4)
 Enter coordinates of vertex V3: (1, 8)
 Enter coordinates of vertex V4: (8, 8)
 Enter coordinates of vertex V5: (8, 1)

Vertices of Polygon (sorted): [ (1, 1) (8, 1) (4, 4) (8, 8) (1, 8) ] #

RESULT: Polygon is not convex.


RAY CASTING

 Enter coordinates of ray point (R): (0, 7)

RESULT: Ray origin outside of polygon.

DONE.
```

Input File for Output 2:

```
5
1,1 4,4 1,8 8,8 8,1
6,7
0,7
```