|  |  |
| --- | --- |
| Name: | Prerna Sunil Jadhav |
| Sap Id: | 60004220127 |
| Class: | T. Y. B. Tech (Computer Engineering) |
| Course: | Advance Algorithm Laboratory |
| Course Code: | DJ19CEL602 |
| Experiment No.: | 07 |

**AIM: Implement Convex Hull using Graham Scan.**

**CODE:**

from functools import cmp\_to\_key

class Point:

    def \_\_init\_\_(self, x = None, y = None):

        self.x = x

        self.y = y

# A global point needed for sorting points with reference

# to the first point

p0 = Point(0, 0)

# A utility function to find next to top in a stack

def nextToTop(S):

    return S[-2]

# A utility function to return square of distance

# between p1 and p2

def distSq(p1, p2):

    return ((p1.x - p2.x) \* (p1.x - p2.x) + (p1.y - p2.y) \* (p1.y - p2.y))

# To find orientation of ordered triplet (p, q, r).

# The function returns following values

# 0 --> p, q and r are collinear

# 1 --> Clockwise

# 2 --> Counterclockwise

def orientation(p, q, r):

    val = ((q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y))

    if val == 0:

        return 0 # collinear

    elif val > 0:

        return 1 # clock wise

    else:

        return 2 # counterclock wise

# A function used by cmp\_to\_key function to sort an array of

# points with respect to the first point

def compare(p1, p2):

    # Find orientation

    o = orientation(p0, p1, p2)

    if o == 0:

        if distSq(p0, p2) >= distSq(p0, p1):

            return -1

        else:

            return 1

    else:

        if o == 2:

            return -1

        else:

            return 1

# Prints convex hull of a set of n points.

def convexHull(points, n):

    # Find the bottommost point

    ymin = points[0].y

    min = 0

    for i in range(1, n):

        y = points[i].y

        # Pick the bottom-most or choose the left

        # most point in case of tie

        if ((y < ymin) or

            (ymin == y and points[i].x < points[min].x)):

            ymin = points[i].y

            min = i

    # Place the bottom-most point at first position

    points[0], points[min] = points[min], points[0]

    # Sort n-1 points with respect to the first point.

    # A point p1 comes before p2 in sorted output if p2

    # has larger polar angle (in counterclockwise

    # direction) than p1

    p0 = points[0]

    points = sorted(points, key=cmp\_to\_key(compare))

    # If two or more points make same angle with p0,

    # Remove all but the one that is farthest from p0

    # Remember that, in above sorting, our criteria was

    # to keep the farthest point at the end when more than

    # one points have same angle.

    m = 1 # Initialize size of modified array

    for i in range(1, n):

        # Keep removing i while angle of i and i+1 is same

        # with respect to p0

        while ((i < n - 1) and

        (orientation(p0, points[i], points[i + 1]) == 0)):

            i += 1

        points[m] = points[i]

        m += 1 # Update size of modified array

    # If modified array of points has less than 3 points,

    # convex hull is not possible

    if m < 3:

        return

    # Create an empty stack and push first three points

    # to it.

    S = []

    S.append(points[0])

    S.append(points[1])

    S.append(points[2])

    # Process remaining n-3 points

    for i in range(3, m):

        # Keep removing top while the angle formed by

        # points next-to-top, top, and points[i] makes

        # a non-left turn

        while((len(S)>1) and (orientation(nextToTop(S),S[-1],points[i])!=2)):

            S.pop()

        S.append(points[i])

    # Now stack has the output points,

    # print contents of stack

    while S:

        p = S[-1]

        print("(" + str(p.x) + ", " + str(p.y) + ")")

        S.pop()

# Driver Code

input\_points = [(0, 3), (1, 1), (2, 2), (4, 4),

                (0, 0), (1, 2), (3, 1), (3, 3)]

points = []

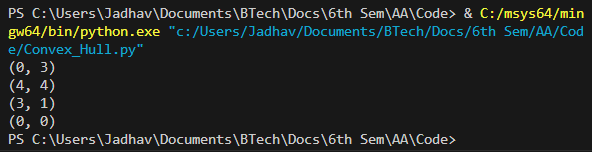
for point in input\_points:

    points.append(Point(point[0], point[1]))

n = len(points)

convexHull(points, n)

**OUTPUT:**

****