Geometría Computacional
Laboratorio Práctica 1
Semestre 2023-2

Profesora: Adriana Ramírez Vigueras[*]
Ayudantes:
Fhernanda Montserrat Romo Olea [**] Marco Antonio Velasco Flores[***]

Fecha de entrega: 18 de marzo de 2023

El objetivo de esta práctica será implementar el algoritmo de **extreme points** para calcular el cierre convexo de un conjunto de puntos.

1. Intrucciones

Uno de los primeros problemas a los que se pueden enfrentar los estudiantes a la hora de implementar algoritmos de geometría computacional es verificar si su solución es correcta.

Para esto pueden crear casos de prueba a mano, meter como entrada estos casos y verificar manualmente si la salida es correcta.

Para facilitar la verificación de su salida y ver si su algoritmo es correcto para una entrada dada sin tener que pasar por el proceso antes mencionado hemos desarrollado una herramienta de visualización de puntos que les ayudara con tal tarea.

Además para mayor flexibilidad se les permite ocupar el lenguaje de programación en el que se sientan más cómodos.

El flujo de trabajo que se espera que usen cuando ocupen la herramienta se puede ver en la grabación del primero laboratorio del curso.

[*] adriana.rv@ciencias.unam.mx
[**] fher@ciencias.unam.mx
[***] mutska@ciencias.unam.mx

1.1. Visualization Helper

La herramienta de visualización está disponible como un binario o instalador junto con la definición de esta práctica.

Deben escoger su binario o instalador según la plataforma en la que se encuentren trabajando.

1. **Linux sistemas basados en debian**: Tienen que utilizar el instalador con extensión *.deb*.
2. **Linux sistemas no basados en debian**: Tienen que utilizar el instalador con extensión *.AppImage*.
3. **Mac OS**: Tienen que utilizar el instalador con extensión *.dmg*.
4. **Windows**: Tienen que utilizar el instalador con extensión *.msi*.

La instalación para Mac y Windows es sencilla, sin embargo para las otras dos opciones pueden necesitar más pasos, si tienen problemas para instalar pueden acercarse al ayudante de laboratorio para asesorarlos.

En todos los casos podría ser que su sistema operativo marque el instalador como malware, esto debido a que ninguno de los instaladores tiene una firma de seguridad embebida aún, pueden ignorar la advertencia.

1.2. Flujo de trabajo

El flujo de trabajo que se espera que ocupen es el siguiente(Se les recomienda ver la grabación de laboratorio 1)

1. Abrir la herramienta de visualización.
2. Generar puntos y acomodarlos a su gusto, se recomienda que no generen demasiados puntos para probar este algoritmo ya que la complejidad es de orden $O(n^4)$.
3. Normalizar los puntos con el primer botón que se encuentra a la derecha del canvas en la herramienta de visualización, esto hace que las coordenadas de todos los puntos se conviertan a enteros y así no tengan que trabajar con números con decimales, pero idealmente deberían ser capaces de hacerlo.
4. Descargar los puntos con el segundo botón que se encuentra a la derecha del canvas y guardarlos como un archivo de texto(.txt)
5. Su programa debe contener una subrutina para leer archivos de texto y así obtener los puntos contenidos en el archivo de texto, más adelante se les explicará en que consiste este formato.
6. Meter como entrada los puntos a su implementación del algoritmo y obtener una solución
7. Su programa debe contener una subrutina para guardar en un archivo de texto **solution.txt** el conjunto de puntos que forman parte del cierre convexo, con cierto formato.
8. Subir su solución(el archivo de texto **solution.txt**) a la herramienta de visualización con el tercer botón que se encuentra a la derecha del canvas. Si su solución es correcta debería dibujarles el cierre convexo del conjunto de puntos, en otro caso revisen su algoritmo.

1.3. Formato

1.3.1. Entrada

Cuando descarguen los puntos como un archivo de texto el formato que verán sera una sola linea de texto de la siguiente manera:

- **1:21:544,2:646:356,3:9:314,4:599:84**.

Esto significa que tenemos cuatro puntos en formato crudo separados por comas:

1. **1:21:544**
2. **2:646:356**
3. **3:9:314**
4. **4:599:84**

A su vez estos cuatro puntos en formato crudo separados por dos puntos representan cuatro puntos, donde el primer número representa el identificador del punto, el segundo número representa su coordenada en x y el tercero su coordenada en y

1. $P_1(21, 544)$
2. $P_2(646, 356)$
3. $P_3(9, 314)$
4. $P_4(599, 84)$

Ustedes deben poder consumir estos puntos como entrada de su algoritmo, a continuación se les da un ejemplo en python que pueden utilizar si ocupan este lenguaje.

```python
from enum import Enum


class Direction(Enum):
    """
    A class used to represent the result
    direction from orientation test
    between three points p, q, r
    """
    COLLINEAR = 1
    CLOCKWISE = 2
    COUNTERCLOCKWISE = 3


class Point:
    """
    A class used to represent a Point

    ...

    Attributes
    ----------
    id : str
        a string number to identify this point
    x : float
        the x-coordinate of the point
    y : float
        the y-coordinate of the point

    """

    def __init__(self, id, x, y):
        """
        Parameters
        ----------
        id : str
            The id of the point
        x : float
            The x-coordinate of the point
        y : float
```

```
            The y-coordinate of the point
        """
        self.id = id
        self.x = x
        self.y = y

    def __str__(self):
        """
        Returns a string representation of this Point
        """
        return f"Point {{ \n id: {self.id} \n x: {self.x} \n y: {self.y} }}"


# Open the points file
contents = []
with open("points.txt", "r", encoding="utf8") as input:
    contents = input.readlines()

# Get the points in  raw format (id:x:y)
raw_points = contents[0].split(',')

points = []

# Create a Point object from each raw point
for raw_point in raw_points:
    id, x, y = raw_point.split(':')
    point = Point(id, float(x), float(y))
    points.append(point)
```

1.3.2. Salida

Cuando tengan los puntos del cierre convexo tienen que guardar su solución en un archivo de texto(**solution.txt** con este nombre). Digamos que su solución son los siguientes puntos

- $P_{13}(21, 544)$
- $P_1(646, 356)$
- $P_4(9, 314)$
- $P_{21}(599, 84)$

Entonces el formato con el que guardan su solución sería el siguiente:

```
13
1
4
21
```

Basta con que pongan el identificador de cada punto por línea. No es necesario que den los puntos en algún orden específico, si quieren probar si todo salio bien suban su archivo solución a la herramienta de visualización.

2. Extreme Points

Para implementar el algoritmo de Extreme Points que consiste en verificar si cada punto está contenido en algún triángulo formado por los puntos restantes tendrán que implementar subrutinas de ayuda.

La primera subrutina que necesitarán y ocuparan para futuras prácticas es un test de orientación.

2.1. Orientation Test

Entrada: p, q, r (tres puntos)
Salida: *Direction* (La dirección de r respecto al segmento pq)
Function `orientation`(p, q, r)**:**
    $valor \longleftarrow (r.y - p.y) * (q.x - p.x) - (q.y - p.y) * (r.x - p.x)$
    si $valor == 0$ **entonces**
        devolver $COLLINEAR$
    fin
    si $valor > 0$ **entonces**
        devolver $COUNTERCLOCKWISE$
    fin
    si $valor < 0$ **entonces**
        devolver $CLOCKWISE$
    fin
End Function

Algoritmo 1: Test de Orientación

2.2. Inside of Triangle Test

La segunda subrutina que necesitan es para verificar si un punto se encuentra contenido dentro un triángulo

Entrada: $p, q, r, current$ (cuatro puntos)
Salida: *Bool* (Si el punto *current* se encuentra dentro del triángulo formado por los puntos p, q, r se regresa $True$, en otro caso $false$)
Function `isPointInsideTriangle`$(p, q, r, current)$**:**
  $orientation_1 \longleftarrow orientation(p, q, current)$
  $orientation_2 \longleftarrow orientation(q, r, current)$
  $orientation_3 \longleftarrow orientation(r, p, current)$
  si $orientation_1 == CLOCKWISE$ ***and***
    $orientation_2 == CLOCKWISE$ ***and***
    $orientation_3 == CLOCKWISE$
   entonces
    devolver $True$
  fin
  si $orientation_1 == COUNTERCLOCKWISE$ ***and***
    $orientation_2 == COUNTERCLOCKWISE$ ***and***
    $orientation_3 == COUNTERCLOCKWISE$
   entonces
    devolver $True$
  fin
  devolver $False$
End Function

Algoritmo 2: Verifica si un punto se encuentra contenido en un triángulo

2.3. Extreme Points

Finalmente con estas subrutinas podemos implementar nuestro algoritmo de **extreme points** que tiene una complejidad de $O(n^4)$:(

Esta es una manera de implementar este algoritmo, lo pueden hacer de otra manera si así lo desean, pero deben respetar la complejidad :(.

Tomen las consideraciones necesarias para los índices, recuerden que normalmente se ocupan índices que empiezan en cero y no en uno.

```
Entrada: points, n (Lista de puntos, Tamaño de la lista de puntos)
Salida: convexHull (Cierre convexo de la lista de puntos)
Function Extreme Points(points, n):
    convexHull ⟵ ∅
    para pointIndex ← 1 to n hacer
        isInside ⟵ False
        para i ← 1 to n hacer
            si i == pointIndex entonces
                continue
            fin
            para j ← i + 1 to n hacer
                si j == pointIndex entonces
                    continue
                fin
                para k ← j + 1 to n hacer
                    si k == pointIndex entonces
                        continue
                    fin
                    p ⟵ points[i]
                    q ⟵ points[j]
                    r ⟵ points[k]
                    current ⟵ points[pointIndex]
                    si isPointInsideTriangle(p, q, r, current) == True entonces
                        isInside ⟵ True
                        break
                    fin
                fin
                si isInside == True entonces
                    break
                fin
            fin
            si isInside == True entonces
                break
            fin
        fin
        si isInside == False entonces
            convexHull.append(points[pointIndex])
        fin
    fin
    devolver convexHull
End Function
```

Algoritmo 3: Extreme Points

3. Consideraciones Y Entregables

- Esta práctica cuenta sobre calificación final(aún por definirse).
- Todo su código debe estar bien documentado.
- Deben dar instrucciones claras y precisas para ejecutar su código en el lenguaje que elijieron.
- No hay prórroga para cambiar la fecha de entrega.

El entregable consiste de un archivo comprimido en zip con su código de solución al problema, instrucciones para ejecutar su código en el lenguaje escogido y un archivo de texto con su nombre completo y número de cuenta.