Taller 2.1

Guía

- La idea de este taller es trabajar sobre los conceptos desarrollados en clase de manera práctica.
- Pueden usar su lenguaje de programación favorito, se dan ejemplos en Python.

En este taller vamos a generar los diferentes tipos de vertices para ser procesados en la lista de eventos.
Con unos puntos de entrada que representan un polígono queremos clasificarlos dentro de los diferentes tipos para ser procesados apropiadamente.

Paso 1: Determinar tipo

Tenemos la siguiente función para determinar qué tipos de vértice tenemos en una secuencia de puntos en $\mathbb{R}^2$ que representa un polígono simple, donde se asume que el último punto conecta con el primer punto.

```
def prod(v,w):
  return v[0]*w[1] -w[0]*v[1]
def minus(v,w):
  return (v[0]-w[0],v[1]-w[1])
#regresa a que tipo de vertice corresponde v2, si v1 y v3 son sus
   vecinos dentro del poligono
#que en sentido antihorario recorre de v1 -> v2 -> v3
def type(v1,v2,v3):
  if min(v1[1],v3[1])>=v2[1]:
    if prod(minus(v1,v2),minus(v3,v2)) >0:
      return 'merge'
    else:
      return 'end'
  elif max(v1[1],v3[1]) <= v2[1]:
    if prod(minus(v1,v2),minus(v3,v2)) >0:
      return 'start'
    else:
      return 'split'
  return 'regular'
```

1: Generar casos de prueba de la función anterior para mirar si se está generando el tipo de vertex adecuado dentro de la secuencia.

2: Corregir la función presentada para que genere los tipos apropiadamente y generar casos de prueba para asegurarse de la funcionalidad apropiada.

Paso 2: Aplicación

Para aplicar el algoritmo de línea de barrido usaremos la estructura de doubly-connected edge list, presentamos el siguiente algoritmo para generar una DCEL a partir de la secuencia de puntos.

```
#funcion auxiliar que genera doubly connected edge list a partir de
#una secuencia de vertices en R^2 donde cada punto en la secuencia
   representa
#el recorrido que sigue la linea borde del poligono en sentido
   antihorario,
#se maneja la convencion de que el primer punto dentro de la lista

class dlcEdge:
    def __init__(self, start, end, next_edge=None):
        self.start = start
        self.end = end
        self.next = next
#esta funcion genera un doubly connected edge, que tiene como next, y en
    sus
#aplicaciones repetidas, la secuencia de puntos en la lista seq, se
   asume que esa
#genera un poligono regular en sentido antihorario, asumimos que len(seq
   )>2
def gen_dcl_de_seq(seq):
  l_seq=len(seq)
  dledge_0=dlcEdge(seq[0], seq[1])
  dledge=dledge_0
  i=1
  while i < l_seq-1:
    dledge.next=dlcEdge(seq[i], seq[i+1])
    dledge=dledge.next
    i+=1
  dledge.next=dlcEdge(seq[l_seq-1], seq[0])
  return dledge_0
#def gen_sew_de_dlc(seq):
dedge_aux=gen_dcl_de_seq(points)
```

Aplicar los respectivos pasos 1 : y 2 : del punto anterior para este algoritmo. En esta secuencia, a través de múltiples aplicaciones de `next` obtenemos el primer `vertex`, y podríamos continuar las secuencia? De lo contrario cómo cambiaríamos el código para lograr esto.

3: Proponer una forma en como se puede guardar la estructura de DCEL para que se pueda ir modificando dentro del algoritmo de línea de barrido. Tener en cuenta que se debe mantener esta estructura durante el algoritmo para que mientras la línea va recorriendo los eventos y actualizando el estado se puedan ir agregando las nuevas diagonales a la estructura de DCEL

4: Construir una función que de acuerdo a la estructura creada el 3 : pueda generar las secuencias correspondientes a los polígonos y-homogeneos a partir de la estructura DCEL.

Paso 3: Ultima preparación para línea de barrido

Qué estructuras necesitamos implementar para aplicar la línea de barrido? Implementar las estructuras correspondientes para al recibir como `input` una secuencia, representando un polígono simple, poder llevar a ejecución el algoritmo.