#!/usr/bin/env python
"""Algoritmo de Fortune"""

from geocomp.common.polygon import Polygon

from random import randint
from geocomp.common import control
from geocomp.common.guiprim import *
from queue import PriorityQueue
from pqdict import pqdict
from geocomp.common.point import Point
from geocomp.common.segment import Segment
from geocomp.voronoi.DCEL import DCEL, Vertex, Hedge, Face
from geocomp.voronoi.BST import BST, get_x_breakpoints, derivada_parabola
from geocomp.voronoi.circumcircle import *
import math

from geocomp import config

class Event():
def **init**(self, point, is_site_event=None, leaf=None, center=None):
self.point = point
self.is_site_event = is_site_event
self.leaf = leaf
self.center = center

```
def __str__(self):
    return f'({self.point.x}, {self.point.y})'
```

class Ordem():
def **init__(self, e): self.x = e.point.x self.y = e.point.y def __eq**(self, other):
return self.y == other.y and self.x == other.x

```
def __lt__(self, other):
    return self.y > other.y or (self.y == other.y and self.x < other.x)

def __gt__(self, other):
    return self.y < other.y or (self.y == other.y and self.x > other.x)
```

def final_plot(leaves, hedges, line_y):

for h in hedges:

h.segment.plot(cor='blue')

```
return [control.plot_parabola(line_y, l.point.x, l.point.y, -20, 20, steps=300) for l in leave
```

def final_unplot(par_plots, hedges):

for par in par_plots:

control.plot_delete(par)

```
for h in hedges:
    h.segment.hide()
```

def event_queue(P):

events = [Event(p, True) for p in P]

Q = pqdict({e : Ordem(e) for e in events}, reverse=False)

return Q

def Fortune(P):

Q = event_queue(P)

V = DCEL()

T = BST()

while Q:

```
        q = Q.pop()
        q.point.hilight()
        sweep = control.plot_horiz_line(q.point.y, color='green')
        if q.is_site_event:
            print(f'({q.point.x}, {q.point.y})', 'evento ponto')
            handle_site_event(q.point, T, Q, V)
        else:
            print(f'({q.point.x}, {q.point.y})', 'evento circulo')
            handle_circle_event(q, T, Q, V)
            q.point.unplot()

        # print('Q:',Q)
        print('T:', T)
        print()
        control.sleep()

        par_plots = final_plot(T.all_leaves(), V.hedges, q.point.y)
        control.sleep()

        final_unplot(par_plots, V.hedges)
        control.plot_delete(sweep)
        q.point.unhilight()

        control.update()
```

```
    print('----------------')

# finalize_voronoi(V, T)
print('fim do Voronoi')
return V
```

def handle_site_event(q, T, Q, V):
if T.is_empty():
T.insert(q)
else:
f = T.search(q)
if f.event is not None:
f.event.point.unplot()
Q.updateitem(f.event, Ordem(Event(Point(math.inf, math.inf))))
Q.pop()
f.event = None

```
    u, f, v = T.split_and_insert(f, q)

    bissect_line = bissect_line_function(u)
    v_1 = V.add_vertex(Point(-200, bissect_line(-200)))
    v_2 = V.add_vertex(Point(200, bissect_line(200)))

    h_12 = Hedge(v_1, v_2)
    V.add_hedge(h_12)
    u.hedge = h_12

    h_21 = Hedge(v_2, v_1)
    V.add_hedge(h_21)
    v.hedge = h_21

    h_12.add_twin(h_21)

    update_events(Q, T, f, f, q)
```

def handle_circle_event(q, T, Q, V):
f = q.leaf
pred, succ, new_node = T.remove(f, Q)

```
left_leaf = new_node.p_i
right_leaf = new_node.p_j

update_events(Q, T, left_leaf, right_leaf, q.point)

u = V.add_vertex(q.center)


pred.hedge.update_origin(u)
x_breakpoints = get_x_breakpoints(pred, q.point.y)
# print(x_breakpoints)
```

```
    # print(q.center.x)
    bissec_pred = bissect_line_function(pred)
    if math.isclose(x_breakpoints[0], q.center.x, rel_tol=1e-6):
        point_pred = Point(200, bissec_pred(200))
    else:
        point_pred = Point(-200, bissec_pred(-200))

    if abs(pred.hedge.dest.p.x) == 200:
        pred.hedge.update_dest(point_pred)


    succ.hedge.update_origin(u)
    x_breakpoints = get_x_breakpoints(succ, q.point.y)
    bissec_succ = bissect_line_function(succ)
    if math.isclose(x_breakpoints[0], q.center.x, rel_tol=1e-6):
        point_succ = Point(200, bissec_succ(200))
    else:
        point_succ = Point(-200, bissec_succ(-200))

    if abs(succ.hedge.dest.p.x) == 200:
        succ.hedge.update_dest(point_succ)

    mid1 = mid_point(left_leaf.point, right_leaf.point)
    slope1 = perp_slope(get_line(left_leaf.point, right_leaf.point))
    # bissect_line = lambda x : slope1*x + mid1.y - mid1.x*slope1
    bissect_line = lambda y : (y - mid1.y)/slope1 + mid1.x

    v = V.add_vertex(Point(bissect_line(-200), -200))
    h_vu = Hedge(v, u)
    V.add_hedge(h_vu)
    new_node.hedge = h_vu

    h_uv = Hedge(u, v)
    V.add_hedge(h_uv)

    h_uv.add_twin(h_vu)
```

```
def is_there_left_triple(leaf):
return leaf.pred is not None and leaf.pred.p_i.pred is not None

def is_there_right_triple(leaf):
return leaf.succ is not None and leaf.succ.p_j.succ is not None

def update_events(Q, T, left_leaf, right_leaf, q):
if is_there_left_triple(right_leaf):
leaf2 = right_leaf.pred.p_i
leaf3 = leaf2.pred.p_i
if leaf2.event is None:
add_circle_event(right_leaf, leaf2, leaf3, q, Q)
```

```

    if is_there_right_triple(left_leaf):
        leaf2 = left_leaf.succ.p_j
        leaf3 = leaf2.succ.p_j
        if leaf2.event is None:
```

```
        add_circle_event(left_leaf, leaf2, leaf3, q, Q)
```

def add_circle_event(leaf1, leaf2, leaf3, q, Q):
p1, p2, p3 = leaf1.point, leaf2.point, leaf3.point
p2.hilight('yellow')
p3.hilight('yellow')
center = circumcenter(p1, p2, p3)
radius = distance(center, p1)
circle = control.plot_circle(center.x, center.y, 'blue', radius)

```
if center.y - radius < q.y or (center.y - radius == q.y and center.x - radius > q.x):
    point = Point(center.x, center.y - radius)
    print('------ cria evento circulo: ', leaf2, f'({center.x}, {center.y})')
    leaf2.event = Event(point, False, leaf2, center)
    Q.additem(leaf2.event, Ordem(leaf2.event))
    point.plot(color='cyan')

control.sleep()
control.plot_delete(circle)
p2.unhilight()
p3.unhilight()
```