Geo1000 – Python Programming – Assignment 4

Due: Friday, October 25, 2019 (18h00)

Introduction

You are expected to create 2 programs and a report. All program files and the report have to be handed in. Start with the files that are distributed via Brightspace. It is sufficient to modify the function definitions inside these files (replace `pass` with your own implementation) — **do not change the function signatures, i.e. their names, which & how many and in which order the functions take arguments**.

This assignment is *preferably* made in groups of 2 (enroll with your group or individually in Brightspace) and the mark you will obtain will count for your final grade of the course. Helping each other is fine. However, make sure that your implementation is your *own*.

This assignment in total can give you 100 points. Your assignment will be marked based on whether your implementation provides a program that runs, does the correct things (as described in the assignment), your code is decent (e.g. use of proper variable names / indentation / etc) and submitted as required (e.g. on time).

It is due: **Friday, October 25, 2019 (18h00)**.

Note that if you submit your assignment after the deadline, points will be removed. For the first day that a submission is late 10 points will be removed before marking. For every day after that another 20 points will be removed. An example: Assume you deliver the assignment at Friday, October 25, 2019, 18h15, the maximum amount of points that then can be obtained for the assignment is 90 $(100 - 10 = 90)$.

Submit the resulting program files (`delaunay.py`, `benchmark.py`) via Brightspace (your last submission will be taken into account, also for determining whether you are late). Upload **a zip file** that contains just the program files and your report **as PDF** (with no folders/hierarchy inside)!

Make sure that each Python file that is handed in, starts with the following comment (augment `Authors` and `Studentnumbers` with your own names and numbers):

```
# GEO1000 - Assignment 4
# Authors:
# Studentnumbers:
```

1 Object-Oriented Delaunay triangulation – delaunay.py – 70 points

Create a program that performs the Delaunay triangulation of a set of random points (see Figure 1).

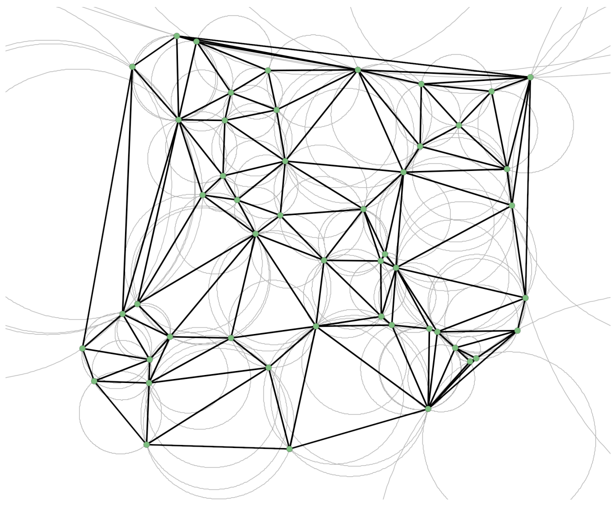

Figure 1: Delaunay triangulation of a set of points. Note that also the circumcircles of the triangles are shown.

> In mathematics and computational geometry, a Delaunay triangulation for a set P of points in a plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid skinny triangles. The triangulation is named after Boris Delaunay for his work on this topic from 1934. — `https://en.wikipedia.org/wiki/Delaunay_triangulation`

The steps you need to make are as follows. See for detailed requirements the docstrings of the skeleton code provided. Implement the:

1. `distance` method of the Point class
2. `__eq__` method of the Point class
3. `covers` method of the Circle class. A circle covers a point if the point is located on the interior or on the border of the circle. Note that for this implementation a circle *also* covers a point if it is very close to its border (so, even if it is slightly outside). With very close we mean that the distance to the border is smaller than $1e{-}8$.
4. `area` / `perimeter` methods of the Circle class
5. `area` / `perimeter` methods of the Triangle class. Note, the area A of a triangle can be computed based on the length of its sides (using Heron's formula):

   $$A = \sqrt{s(s-a)(s-b)(s-c)}$$

   where:

   $$s = \frac{a+b+c}{2}$$

   and a, b and c are the lengths of the sides of the triangle.
6. `circumcircle` method of Triangle class.

   The coordinates of the centre of the circumcircle (u_x, u_y) can be calculated based on the 3 corner points of the triangle (a, b and c) as follows:

   $$u_x = \frac{(a_x^2 + a_y^2)(b_y - c_y) + (b_x^2 + b_y^2)(c_y - a_y) + (c_x^2 + c_y^2)(a_y - b_y)}{D} \qquad (1)$$

$$u_y = \frac{(a_x^2 + a_y^2)(c_x - b_x) + (b_x^2 + b_y^2)(a_x - c_x) + (c_x^2 + c_y^2)(b_x - a_x)}{D} \tag{2}$$

with:

$$D = 2.0\Big(a_x(b_y - c_y) + b_x(c_y - a_y) + c_x(a_y - b_y)\Big) \tag{3}$$

7. boolean method `are_collinear` of DelaunayTriangulation class: This method takes three points (a, b, c) as input, returns True when the result of the orientation test – Equation 4 – is close to zero, False otherwise. With close to zero we mean that the absolute value of the orientation test is smaller than $1e{-}8$.

$$(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y) \tag{4}$$

8. boolean method `is_delaunay` of Delaunay Triangulation. Returns `True` when the circumcircle of the triangle contains no other points than the three points of the triangle itself. For a 3-group of points, check if the 3 points are not collinear. If they are not collinear, take the circumcircle of the triangle and check whether there are exactly three points involved in the triangulation that the circle covers (the three points of the triangle). If the circle covers exactly 3 points, the triangle is Delaunay, otherwise it is not.

9. `triangulate` of Delaunay Triangulation: For all possible 3-group of points that form a triangle: Check whether this triangle conforms to Delaunay criterion. If this is the case, add the triangle to the triangles list.

10. `output_points`, `output_triangles`, `output_circumcircles` of Delaunay Triangulation. The output should look as follows:
    - point per line as well known text string. The first line of the text file should read: `wkt`
    - triangle per line as well known text string, followed by its id, its area and its perimeter, separated by a tab character, first line should read (replace `<tab>` by the correct character):
      `wkt<tab>tid<tab>area<tab>perimeter`
    - circle per line as well known text string, followed by the id of the triangle to which it belongs, its area and its perimeter, separated by a tab character, first line should read (replace `<tab>` by the correct character):
      `wkt<tab>tid<tab>area<tab>perimeter`

Start from the following skeleton:

```
# GEO1000 - Assignment 4
# Authors:
# Studentnumbers:

import math


class Point(object):
    def __init__(self, x, y):
        """Constructor"""
        self.x = x
        self.y = y

    def __str__(self):
        """Well Known Text of this point
        """
        return "POINT({} {})".format(self.x, self.y)

    def __hash__(self):
        """Allows a Point instance to be used
        (as key) in a dictionary or in a set (i.e. hashed collections)."""
        return hash((self.x, self.y))

    def __eq__(self, other):
        """Compare Point instances for equivalence
        (this object instance == other instance?).

        :param other: the point to compare with
        :type other: Point

        Returns True/False
        """
        # Your implementation here
        pass

    def distance(self, other):
        """Returns distance as float to the *other* point
        (assuming Euclidean geometry)

        :param other: the point to compute the distance to
        :type other: Point
        """
        # Your implementation here
        pass


class Circle(object):
    def __init__(self, center, radius):
        """Constructor"""
        self.center = center
        self.radius = float(radius)

    def __str__(self):
        """Returns WKT str, discretizing the circle into straight
        line segments
        """
        N = 400  # the number of segments
        step = 2.0 * math.pi / N
        pts = []
        for i in range(N):
            pts.append(Point(self.center.x + math.cos(i * step) * self.radius,
                             self.center.y + math.sin(i * step) * self.radius))
        pts.append(pts[0])
        coordinates = ["{0} {1}".format(pt.x, pt.y) for pt in pts]
        coordinates = ", ".join(coordinates)
        return "POLYGON(({0}))".format(coordinates)

    def covers(self, pt):
        """Returns True when the circle covers point *pt*,
        False otherwise

        Note that we consider points that are near to the boundary of the
        circle also to be covered by the circle(arbitrary epsilon to use: 1e-8).
        """
        # Your implementation here
        pass

    def area(self):
        """Returns area as float of this circle
        """
        # Your implementation here
        pass

    def perimeter(self):
        """Returns perimeter as float of this circle
        """
        # Your implementation here
        pass


class Triangle(object):
    def __init__(self, p0, p1, p2):
        """Constructor

        Arguments: p0, p1, p2 -- Point instances
        """
        self.p0, self.p1, self.p2 = p0, p1, p2

    def __str__(self):
        """String representation
        """
        points = ["{0.x} {0.y}".format(pt) for pt in (
            self.p0, self.p1, self.p2, self.p0)]
        return "POLYGON(({0}))".format(", ".join(points))

    def circumcircle(self):
        """Returns Circle instance that intersects the 3 points of the triangle.
        """
        # Your implementation here
        pass

    def area(self):
        """Area of this triangle, using Heron's formula."""
        # Your implementation here
        pass

    def perimeter(self):
        """Perimeter of this triangle (float)"""
        # Your implementation here
        pass


class DelaunayTriangulation(object):
    def __init__(self, points):
        """Constructor"""
        self.triangles = []
        self.points = points

    def triangulate(self):
        """Triangulates the given set of points.

        This method takes the set of points to be triangulated
        (with at least 3 points) and for each 3-group of points instantiates
        a triangle and checks whether the triangle conforms to Delaunay
        criterion. If so, the triangle is added to the triangle list.

        To determine the 3-group of points, the group3 function is used.

        Returns None
        """
        # pre-condition: we should have at least 3 points
        assert len(self.points) > 2
        # Your implementation here
        pass

    def is_delaunay(self, tri):
        """Does a triangle *tri* conform to the Delaunay criterion?

        Algorithm:

        Are 3 points of the triangle collinear?
            No:
                Get circumcircle
                Count number of points inside circumcircle
                if number of points inside == 3:
                    Delaunay
                else:
                    not Delaunay
            Yes:
                not Delaunay

        Arguments:
            tri -- Triangle instance
        Returns:
            True/False
        """
        # Your implementation here
        pass

    def are_collinear(self, pa, pb, pc):
        """Orientation test to determine whether 3 points are collinear
        (on straight line).

        Note that we consider points that are nearly collinear also to be on
        a straight line (arbitrary epsilon to use: 1e-8).

        Returns True / False
        """
        # Your implementation here
        pass

    def output_points(self, open_file_obj):
        """Outputs the points of the triangulation to an open file.
        """
        # Your implementation here
        pass

    def output_triangles(self, open_file_obj):
        """Outputs the triangles of the triangulation to an open file.
        """
        # Your implementation here
        pass

    def output_circumcircles(self, open_file_obj):
        """Outputs the circumcircles of the triangles of the triangulation
        to an open file
        """
        # Your implementation here
        pass


def group3(N):
    """Returns generator with 3-tuples with indices to form 3-groups
    of a list of length N.

    Total number of tuples that is generated: N! / (3! * (N-3)!)

    For N = 3: [(0, 1, 2)]
    For N = 4: [(0, 1, 2), (0, 1, 3), (0, 2, 3), (1, 2, 3)]
    For N = 5: [(0, 1, 2), (0, 1, 3), (0, 1, 4), (0, 2, 3),
                (0, 2, 4), (0, 3, 4), (1, 2, 3), (1, 2, 4),
                (1, 3, 4), (2, 3, 4)]

    Example use:

        >>> for item in group3(3):
        ...     print item
        ...
        (0, 1, 2)
    """
    # See for more information about generators for example:
    # https://jeffknupp.com/blog/2013/04/07/improve-your-python-yield-and-
        generators-explained/
    for i in range(N - 2):
        for j in range(i+1, N - 1):
            for k in range(j+1, N):
                yield (i, j, k)


def make_random_points(n):
    """Makes n points distributed randomly in x,y between [0,1000]

    Note, no duplicate points will be created, but might result in slightly
    less than the n number of points requested.
    """
    import random
    pts = list(set([Point(random.randint(0, 1000),
                          random.randint(0, 1000)) for i in range(n)]))
    return pts


def main(n):
    """Perform triangulation of n points and write the resulting geometries
    to text files.
    """
    pts = make_random_points(n)
    dt = DelaunayTriangulation(pts)
    dt.triangulate()
    # using the with statement, we do not need to close explicitly the file
    with open("points.wkt", "w") as fh:
        dt.output_points(fh)
    with open("triangles.wkt", "w") as fh:
        dt.output_triangles(fh)
    with open("circumcircles.wkt", "w") as fh:
        dt.output_circumcircles(fh)


def _test():
    # If you want, you can write tests in this function
    pass


if __name__ == "__main__":
    main(5)
```

Do not use any other imports, than the ones given.

2 Runtime analysis `benchmark.py` and report – 30 points

Together with your code, hand in a report that describes your analysis of the runtime behaviour of the Delaunay triangulation you have created. Your report should not be longer than 2 A4 pages. With your report you answer the question: how does the runtime of your implementation behave, when the number of points that is triangulated increases?

Before you start, read Appendix B of Think Python: How to think like a computer scientist (the book we are using).

Your report should contain a graph that shows the runtime of your implementation of 5, 10, 50, 100, 150, 200 and 250 points being triangulated.

For obtaining the runtime results, implement the function `benchmark` in the given `benchmark` module and use the `time.clock` function. For graphing the results of the benchmark you can use `matplotlib`.

Give in your report an extrapolation for the expected runtime for 500, 1000 and 5000 points based on this graph and your theoretical analysis and describe how you carried out the extrapolation. Note that you could use `numpy` for the extrapolation, if you want.

Hand in your report as PDF file. Name your file as follows: `surname_studentnumber.pdf`.

The report can give you 20 points, the benchmark implementation 10 points.

Start from the following skeleton:

```
# GEO1000 - Assignment 4
# Authors:
# Studentnumbers:

import time
import delaunay


def benchmark():
    # Your implementation here
    pass


if __name__ == "__main__":
    benchmark()
```

For the implementation of the `benchmark`, you are allowed to use `numpy` and `matplotlib`, if you want (both have to be installed on your machine if you want to use them).

Additional literature you can consult:

https://tudelft.on.worldcat.org/oclc/869832329?databaseList=1697,2572,638 – GIS fundamentals, 2nd edition by Steve Wise (chapter 6) – should be accessible from within the TU Delft network using your NetID.