Geo1000 – Python Programming – Assignment 3

Due: Monday, October 14, 2024 (18h00)

Introduction

You are expected to create 1 Python program, consisting of 4 source files. All program files 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. This means that you keep their names, which (data types) & how many arguments the functions take and in which order the functions take arguments as given. Also, in case of a fruitful function, make sure to return the values with their expected type.

This assignment is *preferably* made in groups of 2 (enroll with your group or individually in Brightspace), and your group mark will count for your individual final grade of the course. Helping each other from different groups 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 does the correct things (as described in the assignment), whether your code is decent (e.g. use of proper variable names, indentation, etc), and whether your files are submitted as required (e.g. on time).

It is due: **Monday, October 14, 2024 (18h00)**.

Note that if you submit your assignment after the deadline, some 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 Monday, October 14, 2024, 18h15, the maximum amount of points that then can be obtained for the assignment is 90 $(100 - 10 = 90)$.

Submit the resulting program files (`geometry.py`, `strips.py`, `reader.py`, `query.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 (with no folders/hierarchy inside)!

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

```
# GEO1000 - Assignment 3
# Authors:
# Studentnumbers:
```

1 A program for querying a given set of points

In this assignment you will be given a file with points (with x- and y-coordinates). The program you make should report on the points that overlap a query shape (point, circle, or axis-aligned rectangle) in 2D that is given by the user.

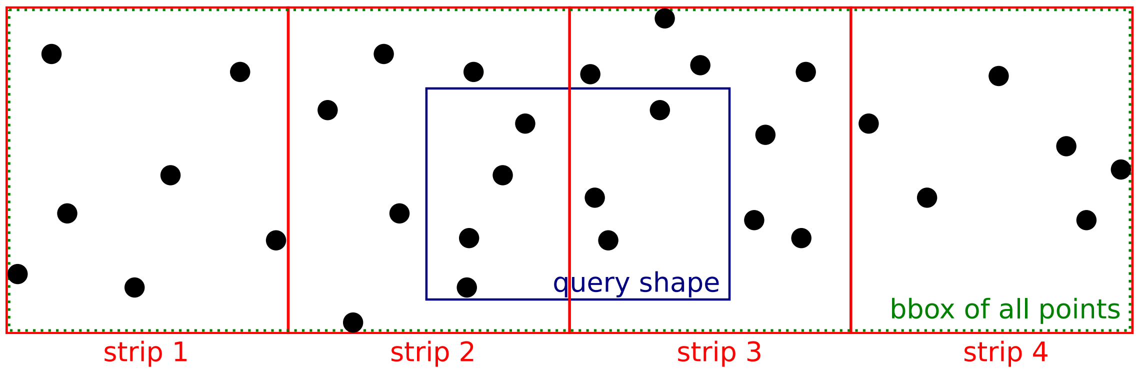

Figure 1: In this example only the points associated with strip 2 and strip 3 have to be visited for checking whether they intersect the query shape.

Because the input file (`points1.txt`) contains so many points, you do not want your program to check *all* points for every user query. Therefore you will implement a simple spatial index structure. For this, your program should first store the points in so called strips. Strips are a set of non-overlapping rectangles. Every strip runs in y-direction from bottom to top of the dataset and has a certain width (which is the same for all strips in the structure). The user can influence — when the file of points is read — how many strips should be used to cover the whole domain of the dataset (and thus influences the width of a strip). Figure 1 gives an illustration.

The program requirements are as follows:

The program asks the user which file to read and how many strips will be used for the strip structure. It then reads the coordinates of the points from the given file. Make sure that lines in the file that start with a '`#`' are skipped. The first uncommented line in the file gives a bounding box (an axis-aligned rectangle in 2D that fits around all the points), which you can use to determine the width of every strip. Add every point that you read from the file to the correct strip in the strip structure. Now the program asks the user for a query shape. Then the program finds all the strips that overlap with the query shape and checks each of the points in these strips if they intersect with the query shape. In this way, the program does not explicitly check all the points from the file every time!

Subsequently the program prints:

- The actual number of points intersecting the query shape in 2D given by the user. A point that is on the border of the query shape is also considered to be inside.
- The leftmost point in the query shape (its WKT[1] string and its identity given by the `id` function[2]). If there are multiple leftmost points, select the one with the lowest y-coordinate.
- The rightmost point in the query shape using its WKT string and its identity given by the `id` function. If there are multiple rightmost points, select the one with the highest y-coordinate.

After the program has printed these statistics, the user can do the following things:

- Write 'quit' followed by enter to quit the program
- Write 'help' to get instructions
- Give another shape to perform another query
- Open another file with points

[1]WKT: Well-known text; see `https://en.wikipedia.org/wiki/Well-known_text`.
[2]`id()` is a built-in function; see `https://docs.python.org/3.7/library/functions.html#id`.

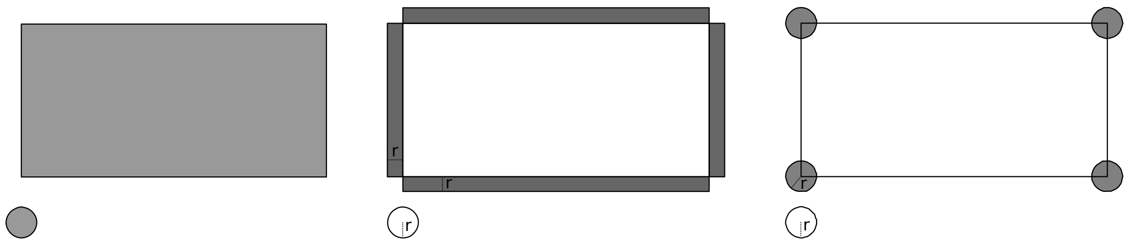

Figure 2: Left: Does this circle intersect this rectangle? Middle/Right: Using additional Circle and Rectangle objects to implement a Circle intersects Rectangle test.

2 Hints

Hint 1: The steps you have to take:

1. Implement all missing methods in `geometry.py`. All `intersects` methods check whether a given shape (point, circle, or axis-aligned rectangle) has any intersection with the interior or boundary of another instance (point, circle, or axis-aligned rectangle). These methods should use type-based dispatch (see §17.8 of the book). Note that you do not have to make 9 different implementations as the intersecting results are commutative/symmetric (e.g. when a `Point` intersects a `Rectangle`, your program should give the exact same result as when the `Rectangle` intersects the `Point`).

   First implement how a point intersects a point, circle, or axis-aligned rectangle. Then implement how two circles intersect and how two axis-aligned rectangles intersect. Then, implement the intersects method for an axis-aligned rectangle and a circle. Inside this particular intersects method, you can instantiate 4 additional rectangles and 4 additional circles to derive whether a circle and a rectangle intersect each other, see Figure 2. Note that the shape of these additional objects depends on the size of the circle object and the shape of the rectangle object that are used for the test (how?).

   Extend the `_test` method with `assert` statements to test whether your implementation functions correctly! Note that you can even do this before having implemented them all. Then they can help you with your implementation. Try to catch all possible cases with your tests.

2. Implement the missing methods in `strips.py`. Start with the constructor, then implement the method `find_overlapping_strips`, and then the remaining ones.

3. Implement the read function in `reader.py`.

4. Implement the missing functions in `query.py`. You should then have a fully working program.

Hint 2: Read the docstrings of the Methods/Functions, these define what exactly should be implemented.

Hint 3: Pay attention to how you deal with points that are falling exactly on the seam of two strips; such points should be stored in the strip with the smallest x-coordinate.

Hint 4: Instead of using the large file with points you are advised to test with the smaller file with less points (`points2.txt`) for testing while developing the `query.py` program.

Hint 5: The `dump` function inside `reader.py` may be helpful for inspection (writing the contents to a textfile, which can be read into a QGIS DelimitedTextLayer).

3 Sample run of `query.py`

```
Welcome to query.py.
============================================================================

Commands available:
-------------------
General:
    help
    quit

Reading points in a structure, defining how many strips should be used:
    open <filenm> into <number_of_strips>

Querying:
    with a point:     p <px> <py>
    with a circle:    c <cx> <cy> <radius>
    with a rectangle: r <llx> <lly> <urx> <ury>
your command>>>
open points2.txt into 5
5 strips
#1 with 33 points, ll: POINT (0.0 0.0), ur: POINT (2.0 10.0)
#2 with 22 points, ll: POINT (2.0 0.0), ur: POINT (4.0 10.0)
#3 with 22 points, ll: POINT (4.0 0.0), ur: POINT (6.0 10.0)
#4 with 22 points, ll: POINT (6.0 0.0), ur: POINT (8.0 10.0)
#5 with 22 points, ll: POINT (8.0 0.0), ur: POINT (10.0 10.0)
your command>>>
p 5.0 5.0
+--------------+
+ Result       +
+--------------+
1 point(s)
leftmost: POINT (5.0 5.0) id: 140173547195728
rightmost: POINT (5.0 5.0) id: 140173547195728
your command>>>
c 10.0 10.0 1.0
+--------------+
+ Result       +
+--------------+
3 point(s)
leftmost: POINT (9.0 10.0) id: 140173547211216
rightmost: POINT (10.0 10.0) id: 140173547211856
your command>>>
r 2.0 2.0 8.0 4.0
+--------------+
+ Result       +
+--------------+
21 point(s)
leftmost: POINT (2.0 2.0) id: 140173547193360
rightmost: POINT (8.0 4.0) id: 140173547210064
your command>>>
quit
Bye, bye.
```

4 Skeletons to start from

4.1 geometry.py (60 points)

```python
# GEO1000 - Assignment 3
# Authors:
# Studentnumbers:

import math

# __all__ leaves out _test method and only makes
# the classes available for "from geometry import *":
__all__ = ["Point", "Circle", "Rectangle"]


class Point:

    def __init__(self, x, y):
        """Constructor.
        Takes the x and y coordinates to define the Point instance.
        """
        self.x = float(x)
        self.y = float(y)

    def __str__(self):
        """Returns WKT String "POINT (x y)".
        """
        pass

    def intersects(self, other):
        """Checks whether other shape has any interaction with
        interior or boundary of self shape. Uses type based dispatch.

        other - Point, Circle or Rectangle

        returns - True / False
        """
        pass

    def distance(self, other):
        """Returns cartesian distance between self and other Point
        """
        pass


class Circle:

    def __init__(self, center, radius):
        """Constructor.
        Takes the center point and radius defining the Circle.
        """
        assert radius > 0
        assert isinstance(center, Point)
        self.center = center
        self.radius = float(radius)

    def __str__(self):
        """Returns WKT str, discretizing the boundary of the circle
        into straight line segments
        """
        N = 400
        step = 2 * 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 intersects(self, other):
        """Checks whether other shape has any interaction with
        interior or boundary of self shape. Uses type based dispatch.

        other - Point, Circle or Rectangle

        Returns - True / False
        """
        pass


class Rectangle:

    def __init__(self, pt_ll, pt_ur):
        """Constructor.
        Takes the lower left and upper right point defining the Rectangle.
        """
        assert isinstance(pt_ll, Point)
        assert isinstance(pt_ur, Point)
        self.ll = pt_ll
        self.ur = pt_ur

    def __str__(self):
        """Returns WKT String "POLYGON ((x0 y0, x1 y1, ..., x0 y0))"
        """
        pass

    def intersects(self, other):
        """Checks whether other shape has any interaction with
        interior or boundary of self shape. Uses type based dispatch.

        other - Point, Circle or Rectangle

        Returns - True / False
        """
        pass

    def width(self):
        """Returns the width of the Rectangle.

        Returns - float
        """
        pass

    def height(self):
        """Returns the height of the Rectangle.

        Returns - float
        """
        pass


def _test():
    """Test whether your implementation of all methods works correctly.
    """
    pt0 = Point(0, 0)
    pt1 = Point(0, 0)
    pt2 = Point(10, 10)
    assert pt0.intersects(pt1)
    assert pt1.intersects(pt0)
    assert not pt0.intersects(pt2)
    assert not pt2.intersects(pt0)

    c = Circle(Point(-1, -1), 1)
    r = Rectangle(Point(0, 0), Point(10, 10))
    assert not c.intersects(r)
    # Extend this method to be sure that you test all intersects methods!
    # Read Section 16.5 of the book if you have never seen the assert statement


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

4.2 strips.py (20 points)

```
# GEO1000 - Assignment 3
# Authors:
# Studentnumbers:

from geometry import Point, Rectangle


class Strip:
    def __init__(self, rectangle):
        """Constructor. Inits a Strip instance with a Rectangle describing
        its shape and an empty points list.
        """
        self.rect = rectangle
        self.points = []


class StripStructure:
    def __init__(self, extent, no_strips):
        """Constructor. Inits a StripStructure instance with the correct
        number of Strip instances and makes sure that the domain is
        correctly divided over the strips.
        """
        self.strips = []
        # Extend this method,
        # so that the right number of strip objects (with the correct extent)
        # are appended to the strips list
        pass

    def find_overlapping_strips(self, shape):
        """Returns a list of strip objects for which their rectangle intersects
        with the shape given.

        Returns - list of Strips
        """
        pass

    def query(self, shape):
        """Returns a list of points that overlaps the given shape.

        For this it first finds the strips that overlap the shape,
        using the find_overlapping_strips method.

        Then, all points of the selected strips are checked for intersection
        with the query shape.

        Returns - list of Points
        """
        pass

    def append_point(self, pt):
        """Appends a point object to the list of points of the correct strip
        (i.e. the strip the Point intersects).

        For this it first finds the strips that overlap the point,
        using the find_overlapping_strips method.

        In case multiple strips overlap the point, the point is added
        to the strip with the left most coordinate.
        Returns - None
        """
        pass

    def print_strip_statistics(self):
        """Prints:
        * how many strips there are in the structure

        And then, for all the strips in the structure:
        * an id (starting at 1),
        * the number of points in a strip,
        * the lower left point of a strip and
        * the upper right point of a strip.

        Returns - None
        """
        pass

    def dumps_strips(self):
        """Dumps the strips of this structure to a str,
        which (if saved in a text file) can be loaded as
        delimited text layer in QGIS.

        Returns - str
        """
        lines = "strip;wkt\n"
        for i, strip in enumerate(self.strips, start=1):
            t = "{0};{1}\n".format(i, strip.rect)
            lines += t
        return lines

    def dumps_points(self):
        """Dumps the points of this structure to a str,
        which (if saved in a text file) can be loaded as
        delimited text layer in QGIS.

        Returns - str
        """
        lines = "strip;wkt\n"
        for i, strip in enumerate(self.strips, start=1):
            for pt in strip.points:
                t = "{0};{1}\n".format(i, pt)
                lines += t
        return lines
```

4.3 reader.py (10 points)

```
# GEO1000 - Assignment 3
# Authors:
# Studentnumbers:

from geometry import Point, Rectangle, Circle
from strips import StripStructure


def read(file_nm, no_strips):
    """Reads a file with on the first uncommented line a bbox
    (4 numbers separated by a space) and subsequently 0 or more lines with
    points (2 numbers separated by a space) into a Strip Structure.

    If no valid box is found in the input file, it returns None.
    Otherwise a StripStructure with 0 or more points is returned.

    Returns - None or a StripStructure instance
    """
    pass
```

```
def dump(structure, strip_file_nm="strips.wkt", point_file_nm="points.wkt"):
    """Dump the contents of a strip structure to 2 files that can be opened
    with QGIS.

    Returns - None
    """
    with open(strip_file_nm, "w") as fh:
        fh.write(structure.dump_strips())
    with open(point_file_nm, "w") as fh:
        fh.write(structure.dump_points())


def _test():
    """You can use this function to test whether
    the read and dump functions work correctly.
    """
    pass


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

4.4 query.py (10 points)

```
# GEO1000 - Assignment 3
# Authors:
# Studentnumbers:

from reader import read
from geometry import Rectangle, Circle, Point
from os.path import basename


def parse(geom_str):
    """Parse a string into a shape object (Point, Circle, or Rectangle)

    Formats that can be given:
    p <px> <py>
    c <cx> <cy> <r>
    r <llx> <lly> <urx> <ury>

    Returns - Point, Circle, or Rectangle
    """
    pass


def print_statistics(result):
    """Prints statistics for the resulting list of Points of a query

    * Number of points overlapping (i.e. number of points in the list)
    * The leftmost point and its identity given by the id function
    * The rightmost point and its identity given by the id function

    Returns - None
    """
    pass


def print_help():
    """Prints a help message to the user, what can be done with the program.
    """
    helptxt = """
Commands available:
-------------------
General:
    help
    quit

Reading points in a structure, defining how many strips should be used:
    open <filenm> into <number_of_strips>

Querying:
    with a point:      p <px> <py>
    with a circle:     c <cx> <cy> <radius>
    with a rectangle: r <llx> <lly> <urx> <ury>"""
    print(helptxt)

# ============================================================================
# Below are some commands that you may use to test your codes:
# open points2.txt into 5
# p 5.0 5.0
# c 10.0 10.0 1.0
# r 2.0 2.0 8.0 4.0
# ============================================================================


def main():
    """The main function of this program.
    """
    structure = None
    print("Welcome to {0}.".format(basename(__file__)))
    print("=" * 76)
    print_help()
    while True:
        in_str = input("your command>>>\n").lower()
        if in_str.startswith("quit"):
            print("Bye, bye.")
            return
        elif in_str.startswith("help"):
            print_help()
        elif in_str.startswith("open"):
            filenm, nstrips = in_str.replace("open ", "").split(" into ")
            structure = read(filenm, int(nstrips))
            structure.print_strip_statistics()
        elif in_str.startswith("p") or in_str.startswith("c") or in_str.startswith(
            "r"):
            if structure is None:
                print("No points read yet, open a file first!")
            else:
                print_statistics(structure.query(parse(in_str)))


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