collinearPoints

October 18, 2018

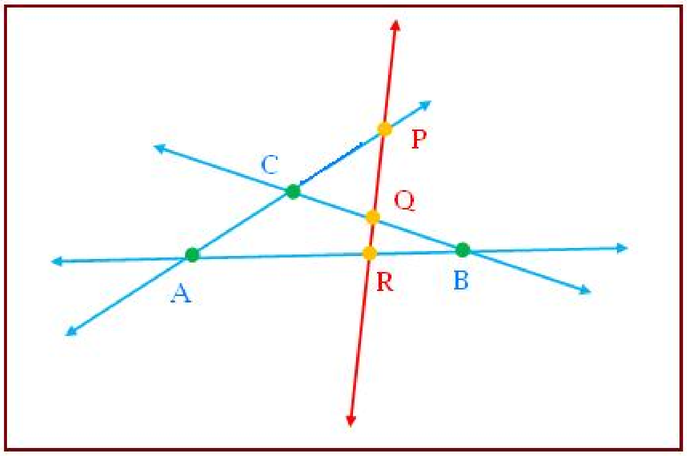
1 Programming Assignment 1: Collinear Points

For this programming assignment, we'll be using a Jupyter notebook.

1.1 Background

1.1.1 Collinear points

Definition of collinearity[1]: In geometry, collinearity of a set of points is the property of their lying on a single line. A set of points with this property is said to be collinear.



Here, points P,Q,R and A,R,B are collinear. However, points A,B,C are non-collinear. For more, refer [2].

1. https://en.wikipedia.org/wiki/Collinearity
2. http://www.mathcaptain.com/geometry/collinear-points.html

1.1.2 Parameterizing lines

In order to determine whether a set of points all lie on the same line we need a standard way to define (or parametrize) a line.

- One way of defining a line is as the set of points (x, y) such that $y = ax + b$ for some fixed real values a, b.
- We call a the **slope** of the line and b is the y-intercept which is defined as the value of y when $x = 0$.

- This parameterization works for *almost* all lines. It does not work for vertical lines. For those lines we define a to be **infinity** and b to be the x intercept of the line (the line is parallel to the y axis so it does not intercept the y axis (other than if it is the vertical line going through the origin).

To summarize, given two different points $(x_1, y_1) \neq (x_2, y_2)$, we define the parameterization (a, b) as: * **if** $x_1 = x_2$**:** (Inf, x_1) * **Else:** (a, b) such that $y_1 = ax_1 + b$ and $y_2 = ax_2 + b$.

1.2 Task

Given an input file with an arbitrary set of co-ordinates, your task is to use pyspark library functions and write a program in python3 to find if three or more points are collinear.

For instance, if given these points: {(1,1), (0,1), (2,2), (3,3), (0,5), (3,4), (5,6), (0,-3), (-2,-2)}

Sets of collinear points are: {((-2,-2), (1,1), (2,2), (3,3)), ((0,1), (3,4), (5,6)), ((0,-3), (0,1), (0,5))}. Note that the ordering of the points in a set or the order of the sets does not matter.

Note:

Every set of collinear points has to have at least three points (any pair of points lie on a line).

There are two types of test cases:

Visible Test cases: Test cases given to you as a part of the notebook. These tests will help you validate your program and figure out bugs in it if any.

Hidden Test cases: Test cases that are not given as a part of the notebook, but will be used for grading. Cells in this notebook that have "##Hidden test cases here" are read-only cells containing hidden tests.

Any cell that does not require you to submit code cannot be modified. For example: Assert statement unit test cells. Cells that have "**# YOUR CODE HERE**" are the ONLY ones you will need to alter.

DO NOT change the names of functions.

Remove the "Raise NotImplementedError()" line when you write the definition of your function.

1.2.1 Description of the Approach

The goal of this assignment is to make you familiar with programming using pyspark. There are many ways to find sets of collinear points from a list of points. For the purposes of this assignment, we shall stick with the below approach:

1. List all pairs of points. You can do that efficiently in spark by computing cartesian product of the list of points with itself. For example, given three points $[(1,0), (2,0), (3,0)]$, we construct a list of nine pairs
   $[((1,0), (1,0)), ((1,0), (2,0)), ((1,0), (3,0))$
   $((2,0), (1,0)), ((2,0), (2,0)), ((2,0), (3,0))$
   $((3,0), (1,0)), ((3,0), (2,0)), ((3,0), (3,0))]$
2. Remove the pairs in which the same point appears twice such as $((2,0), (2,0))$. After these elimination you end up (for this example) with a list of just six pairs:
   $[((1,0), (2,0)), ((1,0), (3,0)), ((2,0), (1,0)), ((2,0), (3,0)), ((3,0), (1,0)), ((3,0), (2,0))]$
3. For each pair of points, find the parameterization (a, b) of the line connecting them as described above.

4. Group the pairs according to their parameters. Clearly, if two pairs have the same (a, b) values, all points in the two pairs lie on the same line.

5. Eliminate the groups that contain only one pair (any pair of points defines a line).

6. In each of the remaining groups, unpack the point-pairs to identify the individual points. Note that if a set of points $(x_1, y_1), \ldots, (x_k, y_k)$ lie on the same line then each point will appear $k-1$ times in the list of point-pairs. You therefore need to transform the list of points into sets to remove duplicates.

7. Output the sets of 3 or more colinear points.

Your task is to implement the described algorithm in Spark. You should use RDD's all the way through and collect the results into the driver only at the end.

1.2.2 Notebook Setup

```
In [ ]: from pyspark import SparkContext, SparkConf

        #We can create a SparkConf() object and use it to initialize the spark context
        conf = SparkConf().setAppName("Collinear Points").setMaster("local[4]") #Initialize spa
        sc = SparkContext(conf=conf)

        from pyspark.rdd import RDD
```

1.2.3 Helper Functions

Here are some helper functions that you are encouraged to use in your implementations. Do not change these functions.

The function format_result takes an element of the form shown below in the example. It outputs a tuple of all points that are collinear (shown below).

Input: ((A,slope), [C1,..., Ck]) where each of A, C1, ..., Ck is a point of form (Ax, Ay) and slope is of type float.

Example Code

```
my_input = (((2, 1), 0.5), [(4, 2), (6, 3)])
format_result(my_input)
```

Output: (C1,..., Ck, A) each of A,C1,...,Ck is a point of form (Ax, Ay)

Example Output

```
((4, 2), (6, 3), (2, 1))
```

Hint : The above example is given just to provide the input and output format. This function is called a different way in the spark exercise.

```
In [ ]: def format_result(x):
            x[1].append(x[0][0])
            return tuple(x[1])
```

```
In [ ]: def to_sorted_points(x):
            """
            Sorts and returns a tuple of points for further processing.
            """
            return tuple(sorted(x))
```

1.3 Exercises

Here are some functions that you will implement. You should follow the function definitions, and use them appropriately elsewhere in the notebook.

1.3.1 Exercise 1: to_tuple

Example The function to_tuple converts each point of form 'Ax Ay' into a point of form (Ax, Ay) for further processing.

Example Code

```
my_input = '2 3'
to_tuple(my_input)
```

Example Output

```
(2, 3)
```

Hint : The above example is given just to provide the input and output format. This function is called a different way in the spark exercise.

Definition

```
In [ ]: ## Insert your answer in this cell. DO NOT CHANGE THE NAME OF THE FUNCTION.
        def to_tuple(x):
            # YOUR CODE HERE
            raise NotImplementedError()
```

Unit Tests

```
In [ ]: assert type(to_tuple('1 1')) == tuple, "Incorrect type: Element returned is not a tupl

In [ ]: assert type(to_tuple('1 1')[0])==int and type(to_tuple('1 1')[1])==int, "Incorrect ele

In [ ]: assert to_tuple('1 1') == (1,1), "Incorrect Return Value: Value obtained does not matc
```

1.3.2 Exercise 2: non_duplicates

Example The function non_duplicates checks if a set of points contains duplicates or not.

Input: Pair (A,B) where A and B are of form (Ax, Ay) and (Bx, By) respectively.

Example Code

```
my_input = ((0,0),(1,2))
non_duplicates(my_input)
```

Output: Returns True if A != B, False otherwise.
Example Output

```
True
```

Hint : The above example is given just to provide the input and output format. This function is called a different way in the spark exercise.

Definition

```
In [ ]: ## Insert your answer in this cell. DO NOT CHANGE THE NAME OF THE FUNCTION.
        def non_duplicates(x):
            """
            Use this function inside the get_cartesian() function to 'filter' out pairs with d
            """
            # YOUR CODE HERE
            raise NotImplementedError()
```

Unit Tests

```
In [ ]: assert type(non_duplicates(((0,0),(1,2)))) == bool, "Incorrect Return type: Function sh
```

```
In [ ]: assert non_duplicates(((0,0),(1,2))) == True, "No duplicates are present"
```

```
In [ ]: assert non_duplicates(((0,0),(0,0))) == False, "Duplicates exist: (0,0)"
```

1.3.3 Exercise 3: get_cartesian

Example The function get_cartesian does a cartesian product of an RDD with itself and returns an RDD with DISTINCT pairs of points.
Input: An RDD containing the given list of points
Output: An RDD containing The cartesian product of the RDD with itself
Example Code

```
test_rdd = sc.parallelize([(1,0), (2,0), (3,0)])
get_cartesian(test_rdd).collect()
```

Example Output

```
[((1, 0), (2, 0)), ((1, 0), (3, 0)), ((2, 0), (1, 0)), ((2, 0), (3, 0)), ((3, 0), (1, 0)), ((3
```

Refer: http://spark.apache.org/docs/latest/api/python/pyspark.html?highlight=cartesian#pyspark.RDD.c

Definition

```
In [ ]: ## Insert your answer in this cell. DO NOT CHANGE THE NAME OF THE FUNCTION.
        def get_cartesian(rdd):
            # YOUR CODE HERE
            raise NotImplementedError()
```

Unit Tests

```
In [ ]: test_rdd = sc.parallelize([(1,0), (2,0), (3,0)])

        l = [((1, 0), (2, 0)), ((1, 0), (3, 0)), ((2, 0), (1, 0)), ((2, 0), (3, 0)), ((3, 0),

        assert isinstance(get_cartesian(test_rdd), RDD) == True, "Incorrect Return type: Functi
        assert set(get_cartesian(test_rdd).collect()) == set(l), "Incorrect Return Value: Value
```

```
In [ ]: ##Hidden test cases here
```

```
In [ ]: ##Hidden test cases here
```

1.3.4 Exercise 4: find_slope

Example The function find_slope computes slope between points A and B and returns it in the format specified below.

Input: Pair (A,B) where A and B are of form (Ax, Ay) and (Bx, By) respectively.

Example Code

```
my_input = ((1,2),(3,4))
find_slope(my_input)
```

Output: Pair ((A,slope), B) where A and B have the same definition as input and slope refers to the slope of the line segment connecting point A and B.

Example Output

```
(((1, 2), 1.0), (3, 4))
```

Note: If Ax == Bx, use slope as "inf".

Hint : The above example is given just to provide the input and output format. This function is called a different way in the spark exercise.

Definition

```
In [ ]: ## Insert your answer in this cell

        def find_slope(x):
            # YOUR CODE HERE
            raise NotImplementedError()
```

Unit Tests

```
In [ ]: assert type(find_slope(((1,2),(3,4)))) == tuple, "Function must return a tuple"
```

```
In [ ]: assert find_slope(((1,2),(-7,-2)))[0][1] == 0.5, "Slope value should be 0.5"
```

```
In [ ]: assert find_slope(((1,2),(3,4))) == (((1,2),1),(3,4)), "Incorrect return value: Value o
```

```
In [ ]: assert find_slope(((1,2),(1,5))) == (((1,2),"inf"),(1,5)), "Incorrect return value: Val
In [ ]: assert find_slope(((1,2),(2,5))) == (((1,2),3),(2,5)), "Incorrect return value: Value
```

```
In [ ]: ##Hidden test cases here
```

```
In [ ]: ##Hidden test cases here
```

```
In [ ]: ##Hidden test cases here
```

1.3.5 Exercise 5: find_collinear

Example The function find_collinear finds the set of collinear points.

Input: An RDD (which is the output of the get_cartesian() function.

Output: An RDD containing the list of collinear points formatted according to the format_result function.

Approach: 1. Find the slope of the line between all pairs of points A = (Ax, Ay) and B = (Bx, By). 2. For each (A, B), find all points C = ((C1x, C1y), (C2x, C2y), … (Cnx, Cny)) where slope of (A,B) = slope of (A, Ci). 3. Return (A, B, Ck) where Ck = all points of C which satisfy the condition 1.

The assert statement unit tests for this function will help you with this. **Hint :** You should use the above helper functions in conjunction with Spark RDD API (refer http://spark.apache.org/docs/latest/api/python/pyspark.html?highlight=rdd#pyspark.RDD) Finally, use helper function format_result() appropriately from inside this function after you have implemented the above operations.

Definition

```
In [ ]: def find_collinear(rdd):
            # YOUR CODE HERE
            raise NotImplementedError()
```

Unit Tests

```
In [ ]: def verify_collinear_sets(collinearpointsRDD, testlist):
            collinearpoints = [tuple(sorted(x)) for x in list(set(collinearpointsRDD.collect()
            testlist = [tuple(sorted(x)) for x in list(set(testlist))]
            return set(collinearpoints) == set(testlist)
```

```
In [ ]: test_rdd = sc.parallelize([((4, 2), (2, 1)), ((4, 2), (-3, 4)), ((4, 2), (6, 3)), ((2,
        assert isinstance(find_collinear(test_rdd), RDD) == True, "Incorrect return type: Func
```

```
In [ ]: assert verify_collinear_sets(find_collinear(test_rdd), [((2, 1), (4, 2), (6, 3))]), "I
```

```
In [ ]: ##Hidden test cases here
```

Unit Tests II : Using the output of get_cartesian(rdd)

```
In [ ]: test_rdd = sc.parallelize([(4, -2), (2, -1), (-3,4), (6,3), (-9,4), (6, -3), (8,-4), (
        test_rdd = get_cartesian(test_rdd)
        assert verify_collinear_sets(find_collinear(test_rdd), [((6, -3), (6, 3), (6, 9)), ((2
```

```
In [ ]: ##Hidden test cases here
```

1.3.6 Exercise 6: The build_collinear_set function

Example Using the above functions that you have written along with pyspark functions, write the **build_collinear_set** function and returns an RDD containing the set of collinear points.
Input: RDD containing the given set of points
Output: RDD containing the set of collinear points
Hint : Remember that the input RDD consists of a set of strings. Remember to pre-process them using the to_tuple function before performing other operations.

Definition

```
In [ ]: def build_collinear_set(rdd):

            # YOUR CODE HERE
            raise NotImplementedError()

            # Sorting each of your returned sets of collinear points. This is for grading purp
            # YOU MUST NOT CHANGE THIS.
            rdd = rdd.map(to_sorted_points)

            return rdd
```

Unit Tests

```
In [ ]: test_rdd = sc.parallelize(['4 -2', '2 -1', '-3 4', '6 3', '-9 4', '6 -3', '8 -4', '6 9
        assert isinstance(build_collinear_set(test_rdd), RDD) == True, "build_collinear_set sho
```

1.3.7 The process function

Definition

```
In [ ]: def process(filename):
            """
            This is the process function used for finding collinear points using inputs from d
            Input: Name of the test file
            Output: Set of collinear points
            """
            # Load the data file into an RDD
            rdd = sc.textFile(filename)

            rdd = build_collinear_set(rdd)

            # Collecting the collinear points RDD in a set to remove duplicate sets of colline
            res = set(rdd.collect())

            return res
```

Unit Tests: Testing the build_collinear_set function using the process function NOTE: You may assume that input files do not have duplicate points.

```
In [ ]: assert process("data.txt") == {((-2, -2), (1, 1), (2, 2), (3, 3)), ((0, 1), (3, 4), (5

In [ ]: assert process("data50.txt") == {((3, 6), (7, 4), (9, 3)), ((1, 6), (3, 6), (4, 6), (7
                                          ((0, 2), (3, 1), (6, 0)), ((1, 0), (2, 0), (5, 0), (6
                                          ((1, 3), (3, 6), (5, 9)), ((0, 8), (4, 6), (6, 5)),
                                          ((6, 0), (6, 1), (6, 5), (6, 9)),
                                          ((7, 2), (7, 3), (7, 4), (7, 6), (7, 8)), ((3, 1), (3
                                          ((0, 2), (1, 2), (5, 2), (7, 2)), ((0, 3), (2, 5), (3
                                          ((0, 2), (1, 3), (2, 4), (4, 6), (5, 7)), ((1, 2), (4
                                          ((0, 3), (4, 6), (8, 9)), ((9, 3), (9, 4), (9, 5)), (
                                          ((0, 5), (2, 4), (4, 3), (8, 1)), ((0, 8), (1, 6), (2
                                          ((3, 6), (5, 2), (6, 0)), ((5, 9), (6, 9), (8, 9)),
                                          ((0, 8), (1, 8), (7, 8)), ((0, 4), (1, 3), (3, 1)), (
                                          ((1, 2), (2, 4), (3, 6)), ((0, 7), (1, 5), (3, 1)),
                                          ((1, 5), (2, 4), (3, 3), (6, 0)), ((0, 2), (3, 3), (9
                                          ((0, 7), (1, 6), (2, 5), (4, 3), (5, 2), (6, 1)),
                                          ((0, 4), (1, 5), (5, 9)), ((1, 5), (3, 6), (5, 7), (7
                                          ((1, 6), (3, 3), (5, 0)), ((3, 6), (4, 3), (5, 0)),
                                          ((1, 2), (4, 5), (7, 8), (8, 9)), ((0, 2), (1, 1), (2
                                          ((3, 3), (4, 5), (5, 7), (6, 9)), ((0, 2), (0, 3), (0
                                          ((2, 0), (4, 3), (8, 9)), ((5, 7), (6, 5), (7, 3), (8
                                          ((5, 0), (6, 1), (7, 2), (9, 4)), ((0, 4), (1, 2), (2
                                          ((1, 1), (3, 1), (6, 1), (8, 1)), ((5, 7), (7, 6), (9
                                          ((0, 4), (2, 4), (7, 4), (9, 4)), ((1, 0), (3, 1), (5
                                          ((2, 0), (3, 3), (4, 6), (5, 9)), ((4, 3), (4, 5), (4
                                          ((1, 0), (4, 3), (6, 5), (7, 6)), ((0, 3), (2, 4), (4
                                          ((1, 6), (4, 5), (7, 4)), ((1, 0), (1, 1), (1, 2), (1
                                          ((0, 3), (1, 3), (3, 3), (4, 3), (7, 3), (9, 3)), ((0
                                          ((0, 7), (3, 6), (6, 5), (9, 4)), ((1, 8), (4, 6), (7
                                          ((0, 5), (3, 3), (6, 1)), ((1, 8), (3, 6), (4, 5), (7
                                          ((1, 2), (3, 1), (5, 0)), ((1, 1), (5, 2), (9, 3)),
                                          ((5, 0), (5, 2), (5, 7), (5, 9)), ((0, 5), (1, 5), (2
                                          ((3, 1), (4, 5), (5, 9)), ((2, 0), (2, 4), (2, 5)), (

In [ ]: ##Hidden test cases here

In [ ]: ##Hidden test cases here

In [ ]: ##Hidden test cases here

In [ ]: ##Hidden test cases here
```