ASSIGNMENT 1: Bay Bridges

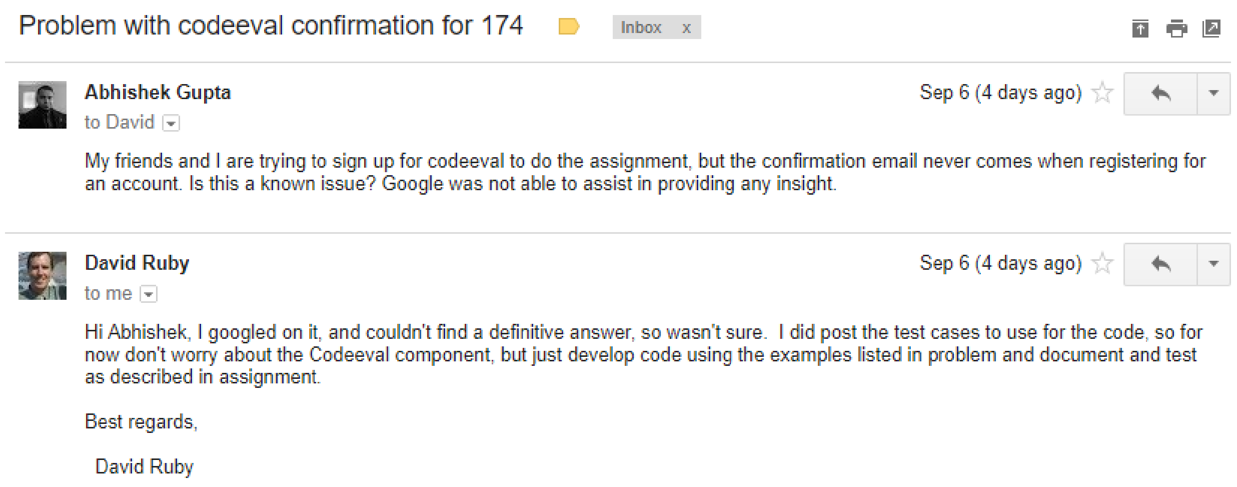
Abhishek Gupta
CSCI 174
15 September 2017

DISCLAIMER

This was a CodeEval assignment. This assignment can be found at the following link:

https://www.codeeval.com/public_sc/109/

Unfortunately, I was not able to open an account so I did not complete or submit the assignment through CodeEval. My correspondence with Dr. Ruby on this matter was the following:



Problem with codeeval confirmation for 174

Abhishek Gupta
to David
Sep 6 (4 days ago)

My friends and I are trying to sign up for codeeval to do the assignment, but the confirmation email never comes when registering for an account. Is this a known issue? Google was not able to assist in providing any insight.

David Ruby
to me
Sep 6 (4 days ago)

Hi Abhishek, I googled on it, and couldn't find a definitive answer, so wasn't sure. I did post the test cases to use for the code, so for now don't worry about the Codeeval component, but just develop code using the examples listed in problem and document and test as described in assignment.

Best regards,

David Ruby

All source and input files along with this documentation / report will be submitted through Blackboard.

I am also interested in volunteering to present my code IF my code is deemed correct. As far as I'm concerned, it is.

Table of Contents

General Problem Description ………………………………………………………………….. 4

Technical Problem Description ……………………………………………………………….. 4

Solution Design ………………………………………………………………………………..….. 5

Sample Input / Output ………………………………………………………………………….. 7

All Submitted Content ……………………………………………………………………….. 12

Works Cited ……………………………………………………………………..……………….. 12

Abhishek Gupta

CSCI 174

15 September 2017

Assignment 1: CodeEval Bay Bridges Challenge

General Problem Description

As stated in the challenge description on CodeEval, there has emerged a novel technological breakthrough to allow the modification of bridges to withstand a 9.5 magnitude earthquake at a significantly reduced cost and time. Rather than fully equipping bridges with this technology (which would take too long and cost too much), we want to only target strategic coordinates of where this technology would prove beneficial.

Technical Problem Description

The goal is to connect as many pairs of points as possible with bridges such that no two bridges cross. When connecting points, you can only connect point 1 with another point 1, point 2 with another point 2, etc.

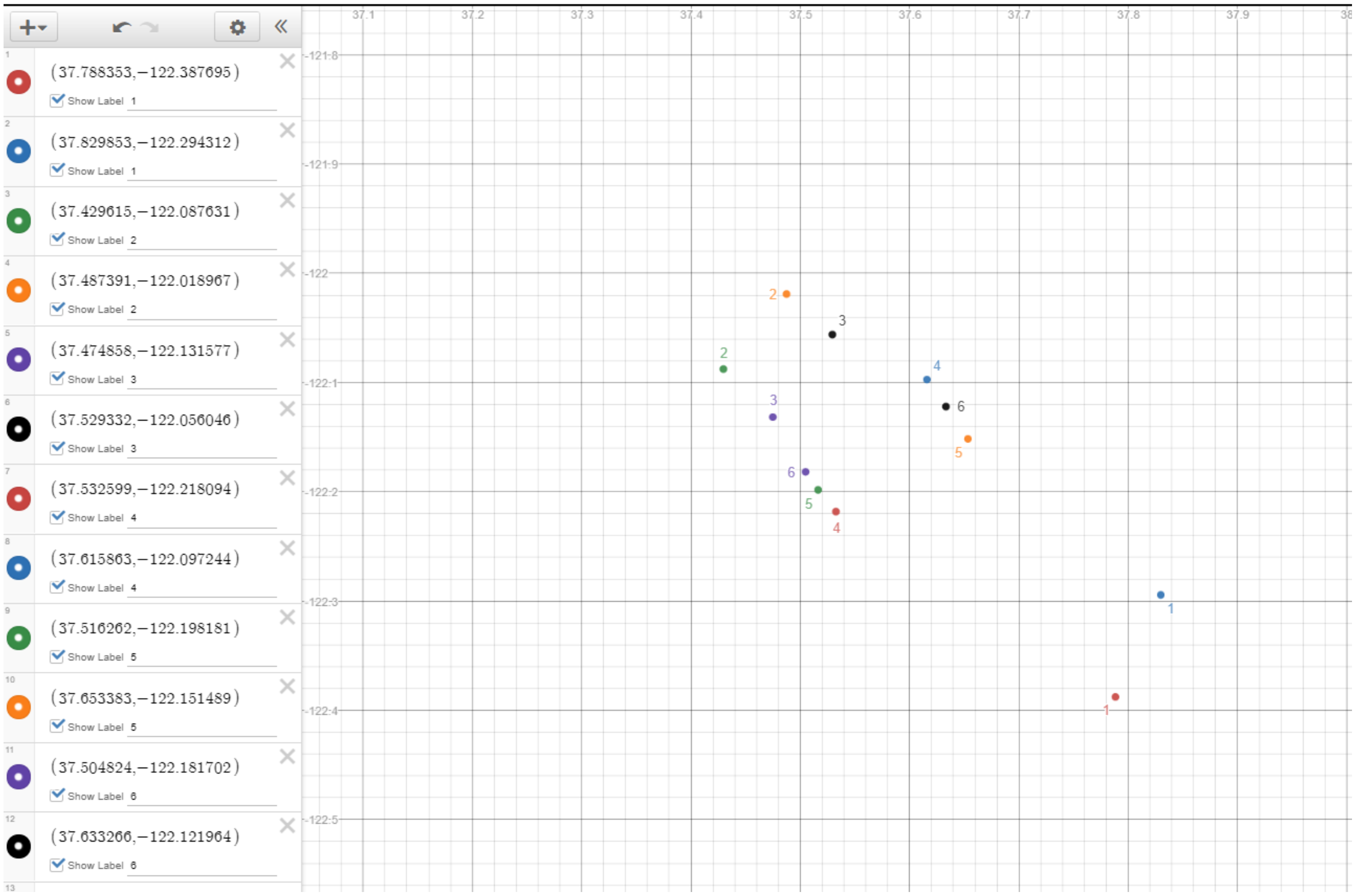
Our program will take in a file as an argument. Inside that file, we will have coordinates that might look like the following:

```
1: ([37.788353, -122.387695], [37.829853, -122.294312]) 2: ([37.429615, -122.087631], [37.487391, -122.018967]) 3:
([37.474858, -122.131577], [37.529332, -122.056046]) 4: ([37.532599,-122.218094], [37.615863,-122.097244]) 5:
([37.516262,-122.198181], [37.653383,-122.151489]) 6: ([37.504824,-122.181702], [37.633266,-122.121964])
```

1, 2, 3, 4, 5, and 6 each represent a line. Once our program finishes reading and processing this txt file, it will output into the console:

```
1 2 3 5 6
```

What this means is that 1, 2, 3, 5, and 6 are the only valid lines because they do not intersect other lines. Why is 4 not included? Because it intersects lines 5 and 6. Let’s take a closer look by graphing the above lines to see what we mean:


1 (37.788353, −122.387695) Show Label 1
2 (37.829853, −122.294312) Show Label 1
3 (37.429615, −122.087631) Show Label 2
4 (37.487391, −122.018967) Show Label 2
5 (37.474858, −122.131577) Show Label 3
6 (37.529332, −122.056046) Show Label 3
7 (37.532599, −122.218094) Show Label 4
8 (37.615863, −122.097244) Show Label 4
9 (37.516262, −122.198181) Show Label 5
10 (37.653383, −122.151489) Show Label 5
11 (37.504824, −122.181702) Show Label 6
12 (37.633266, −122.121964) Show Label 6
13

From the graph above, we see that lines 1, 2, and 3 are valid as they do not intersect other lines. However, line 4 intersects 5 and 6. Since 4 intersects the most lines, it is not valid in our output solution. 5 and 6 are valid once 4 is removed.

Solution Design

Our program must take in a txt file argument with sample input that complies with the formatting provided in "INPUT SAMPLE 1." It must distinguish line labels (1, 2, 3, etc.) and identify that each label has 2 coordinates in the format ([x1, y1], [x2, y2]). The key here is that we don't just want to toss a line out if it has an intersection. Otherwise, 5 and 6 from sample 1 would not be valid lines because they each have an intersection with 4. Rather, we need to find the largest set of lines where there exists no intersection.

Given that the input can have any number of coordinates and lines, we need a way of auto-creating instances of coordinates and lines to deal with any number of them. We should create a Coordinate class that defines what a coordinate is. We will pass in self, x, and y where “self” is pretty much “this” in Java in that it is representing an instance of the object. We will want to do the same for a line which is 2 coordinates. Class Line’s constructor will take in self, index (which indicates where the line is located in a txt file AKA its line number label), x1y1 (representing 1 coordinate), and x2y2. The x1y1 and x2y2 arguments will be Coordinate objects.

The next problem to solve is figuring out how to determine whether two lines intersect. For this, I am using Bryce Boe’s ccw and intersect functions. I have credited him in my Works Cited section. Given two lines, the intersect function will return a boolean to indicate whether or not an intersection between them exists.

We need a function that can go into a file to read, parse, extract information, and store information into a list to be used for further operations. The function should then store that information into Line objects inside of a list. In other words, the function will return a list of the Line objects that contain the contents of the input file minus the brackets, parentheses, commas, and other garbage.

Our setup is complete. Once we have solved the problem of reading the input file and then storing the data that we need from it, we need a function that checks every line and counts the number of intersections that each line has. We cannot simply toss out a line for having an intersection. We need to toss out lines that have the highest number of intersections and we keep going until we have the largest set of lines with no intersections. We will then return this set as a list. We will be using Mr. Boe’s ccw and intersect functions in this check function which will return a count variable of type int.

The above function will then be used in another function that will store all lines without intersections inside of a list ready to be read. The way this will be done is that we will loop through our list of Line objects and we will count how many intersections each line has. If it has 0, then we will append it to the list that we wish to return. In other words, we know that this line is valid so we will push it to our “verified” list. However, as soon as we encounter a line that has 1 or more intersections, we are going to store the number of intersections this line has into a variable (max_intersections) and then store a copy of this line into another variable (max_line). If another line comes along that has a higher intersection count, then we will update max_intersections and update max_line with this new line. After we look through all lines, whatever line is in our max_line variable will get removed.

We will also require a function that will take in the above list, sort it, and then output the results into the console. This will be done via a simple for loop. The sorting could be done using an algorithm we went over in class (such as insertion sort).

My explanations will be far more thorough as comments in my code. Please refer to main.py if you would like a line-by-line explanation.

Sample Input / Output

Sample Input 1 (GIVEN):

```
1: ([37.788353, -122.387695], [37.829853, -122.294312])
2: ([37.429615, -122.087631], [37.487391, -122.018967])
3: ([37.474858, -122.131577], [37.529332, -122.056046])
4: ([37.532599, -122.218094], [37.615863, -122.097244])
5: ([37.516262, -122.198181], [37.653383, -122.151489])
6: ([37.504824, -122.181702], [37.633266, -122.121964])
```

Sample Input 1 Graph:

[SEE PAGE 2]

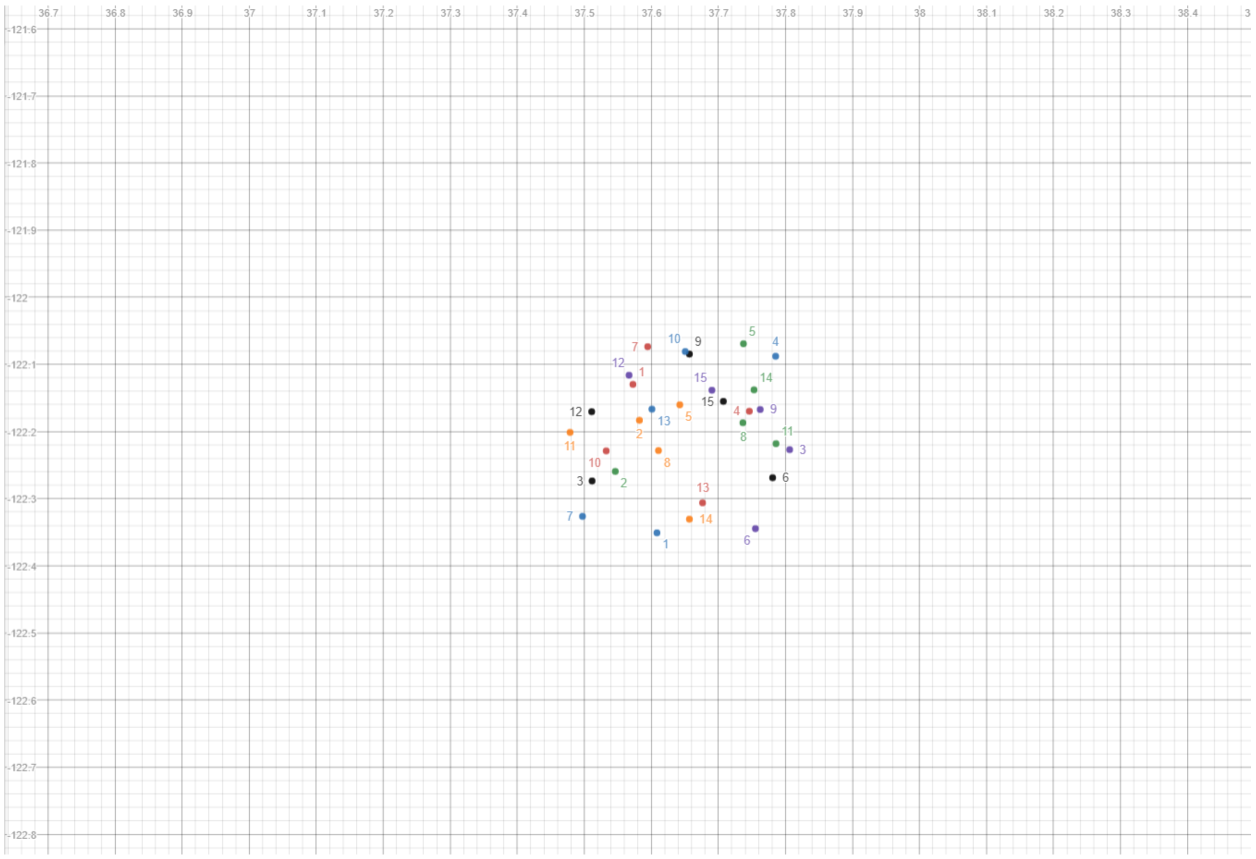
Sample Output 1 (GIVEN):

```
1 2 3 5 6
```

Sample Input 2 (GIVEN):

```
1: ([37.572563, -122.129760], [37.608392, -122.350898])
2: ([37.546241, -122.259403], [37.582266, -122.183210])
3: ([37.806409, -122.227005], [37.511585, -122.273610])
4: ([37.746237, -122.169757], [37.785464, -122.087857])
5: ([37.737455, -122.069225], [37.642475, -122.160176])
6: ([37.755297, -122.344646], [37.780991, -122.268794])
7: ([37.594566, -122.073618], [37.497324, -122.326342])
8: ([37.736614, -122.186938], [37.610637, -122.228337])
9: ([37.762481, -122.167198], [37.656783, -122.084612])
10: ([37.532676, -122.228831], [37.650623, -122.080848])
11: ([37.786019, -122.218078], [37.478787, -122.201259])
12: ([37.566752, -122.116095], [37.511017, -122.170461])
13: ([37.676436, -122.306188], [37.600907, -122.166662])
14: ([37.753226, -122.137899], [37.656818, -122.330516])
15: ([37.690402, -122.138457], [37.707493, -122.155059])
```

Sample Input 2 Graph:



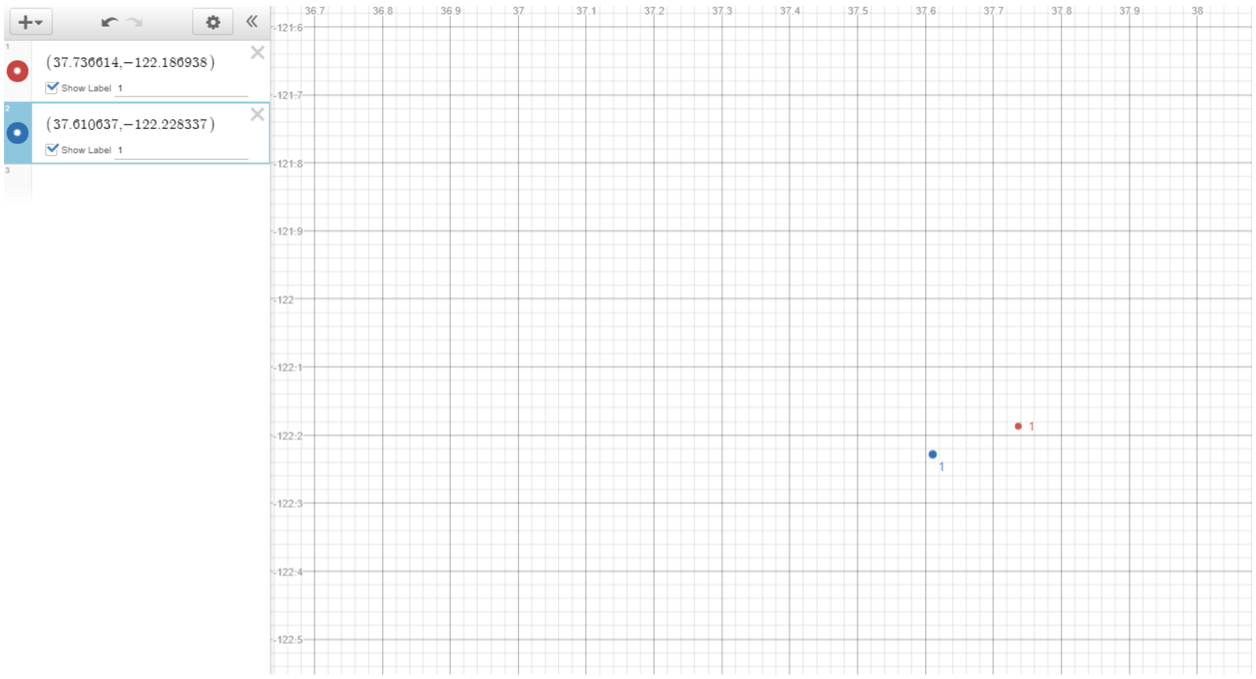
Sample Output 2:

```
2 3 4 5 6 8 10 12 15
```

Sample Input 3:

1: ([37.736614, -122.186938], [37.610637, -122.228337])

Sample Input 3 Graph:



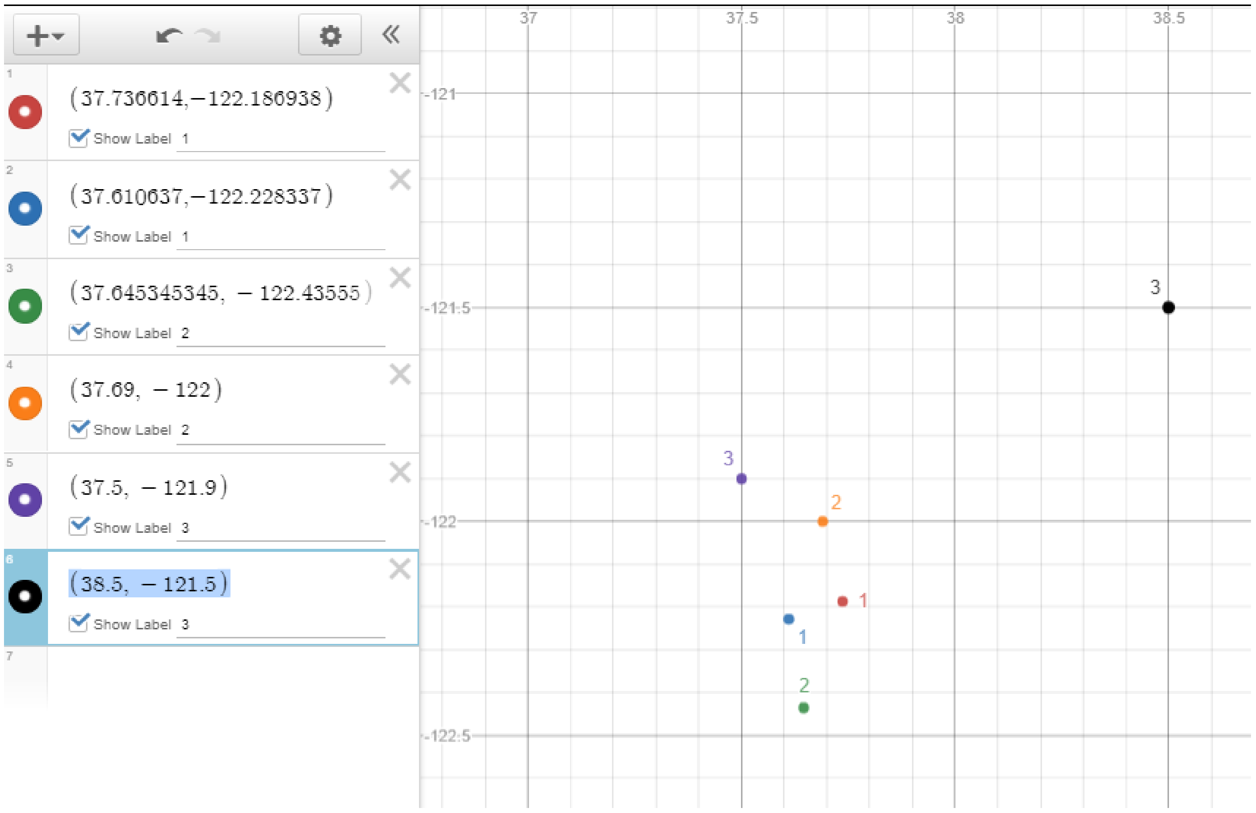
Sample Output 3:

1

Sample Input 4:

1: ([37.736614, -122.186938], [37.610637, -122.228337])
2: ([37.645345345, -122.43555], [37.69, -122])
3: ([37.5, -121.9], [38.5, -121.5])

Sample Input 4 Graph:



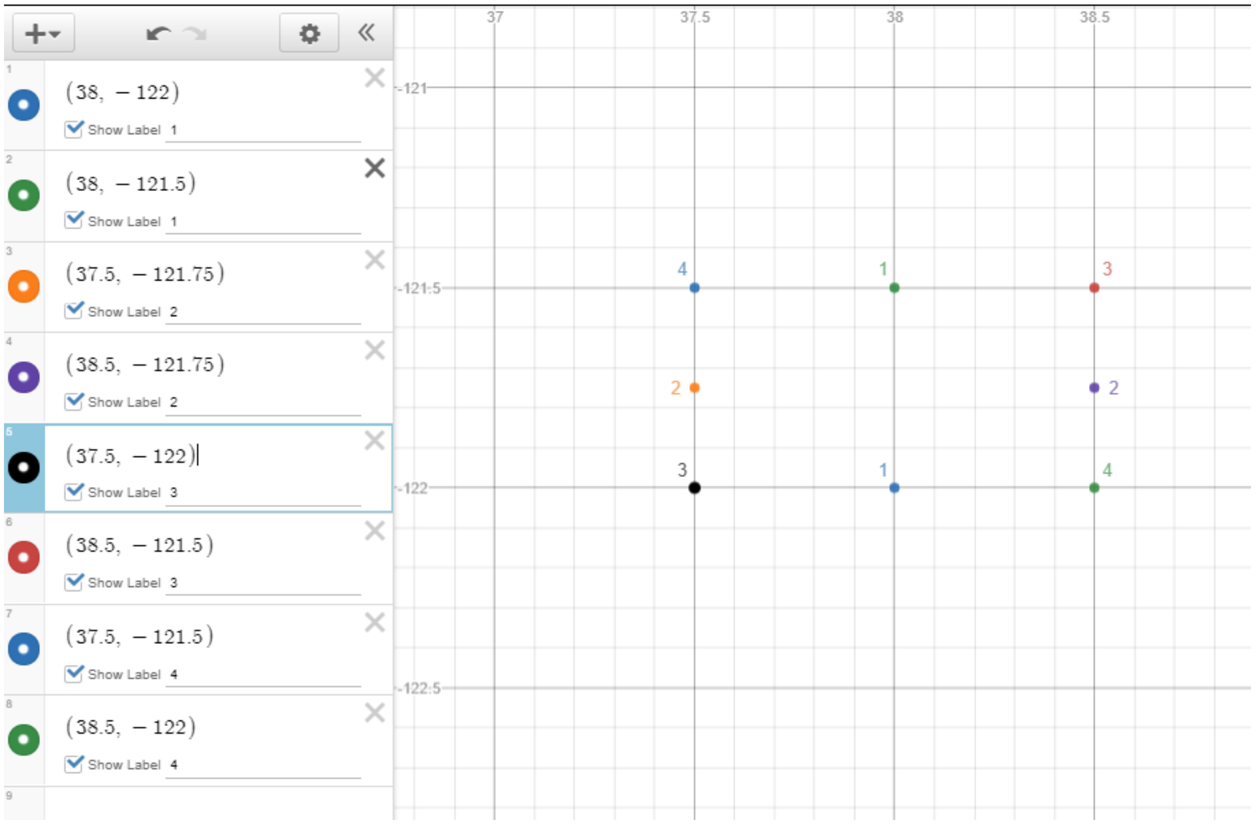
Sample Output 4:

1 3

Sample Input 5:

1: ([38, -122], [38, -121.5])
2: ([37.5, -121.75], [38.5, -121.75])
3: ([37.5, -122], [38.5, -121.5])
4: ([37.5, -121.5], [38.5, -122])

Sample Input 5 Graph:



(38, −122) Show Label 1
(38, −121.5) Show Label 1
(37.5, −121.75) Show Label 2
(38.5, −121.75) Show Label 2
(37.5, −122) Show Label 3
(38.5, −121.5) Show Label 3
(37.5, −121.5) Show Label 4
(38.5, −122) Show Label 4

Sample Output 5:

1

Sample Input 6:

[blank text file]

Sample Output 6:

Input file is blank.

All Submitted Content

- main.py : The actual source code for the problem.
- Abhishek Gupta ASG1 Report.pdf : This document.
- A series of input files. The first two are given.

Works Cited

1. **Article:** Line Segment Intersection Algorithm (Article + Comments Section)
   http://bryceboe.com/2006/10/23/line-segment-intersection-algorithm/
   Purpose: Shows code on how to determine if 2 lines intersect.
2. **Article:** The Key to Solving CodeEval's Bay Bridge Problem
   http://deepinthecode.com/2014/03/26/key-solving-codeevals-bay-bridges-challenge/
   Purpose: Does a better job in explaining what problem we are trying to solve.
3. **Article:** Bay Bridges Challenge (solved!)
   https://shashankr.wordpress.com/2014/03/10/bay-bridges-challenge-solved/
   Purpose: Goes over the problem in depth. Provides a mathematical proof for the solution.
4. **GitHub:** lucasrangit/CodeEval/bridges.py
   https://github.com/lucasrangit/CodeEval/blob/master/bridges.py
   Purpose: I took Lucas Magasweran's code that was done in Python 2 and converted it into Python 3. I reformatted the file, made improvements, removed unnecessary / confusing bits of code, added in some of my own code, and then renamed / rearranged / added the variables / functions / classes to make the code easier to understand. Then I heavily commented the code. Bryce Boe's ccw and intersect functions were also included.
5. **Stack Overflow:** What does if __name__ == "__main__": do?
   https://stackoverflow.com/questions/419163/what-does-if-name-main-do
6. **YouTube:** Insertion Sort in Python
   Purpose: I used this person's code and implemented it into my own code. I wanted to use something we learned in class so I picked Insertion Sort to sort my output.
   https://www.youtube.com/watch?v=lEA31vHiry4