**ASSIGNMENT 1: Bay Bridges**

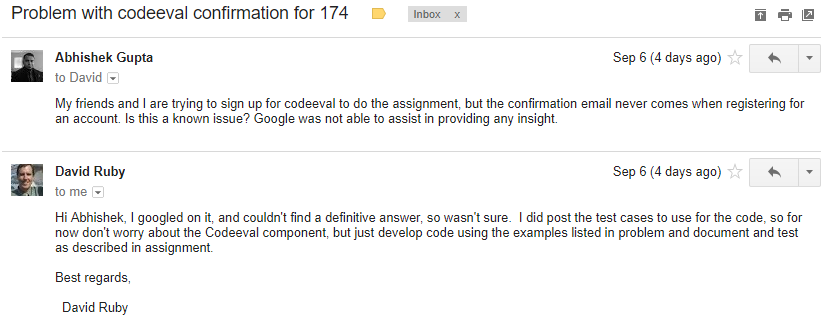
Abhishek Gupta  
CSCI 174  
15 September 2017

**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:



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

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.

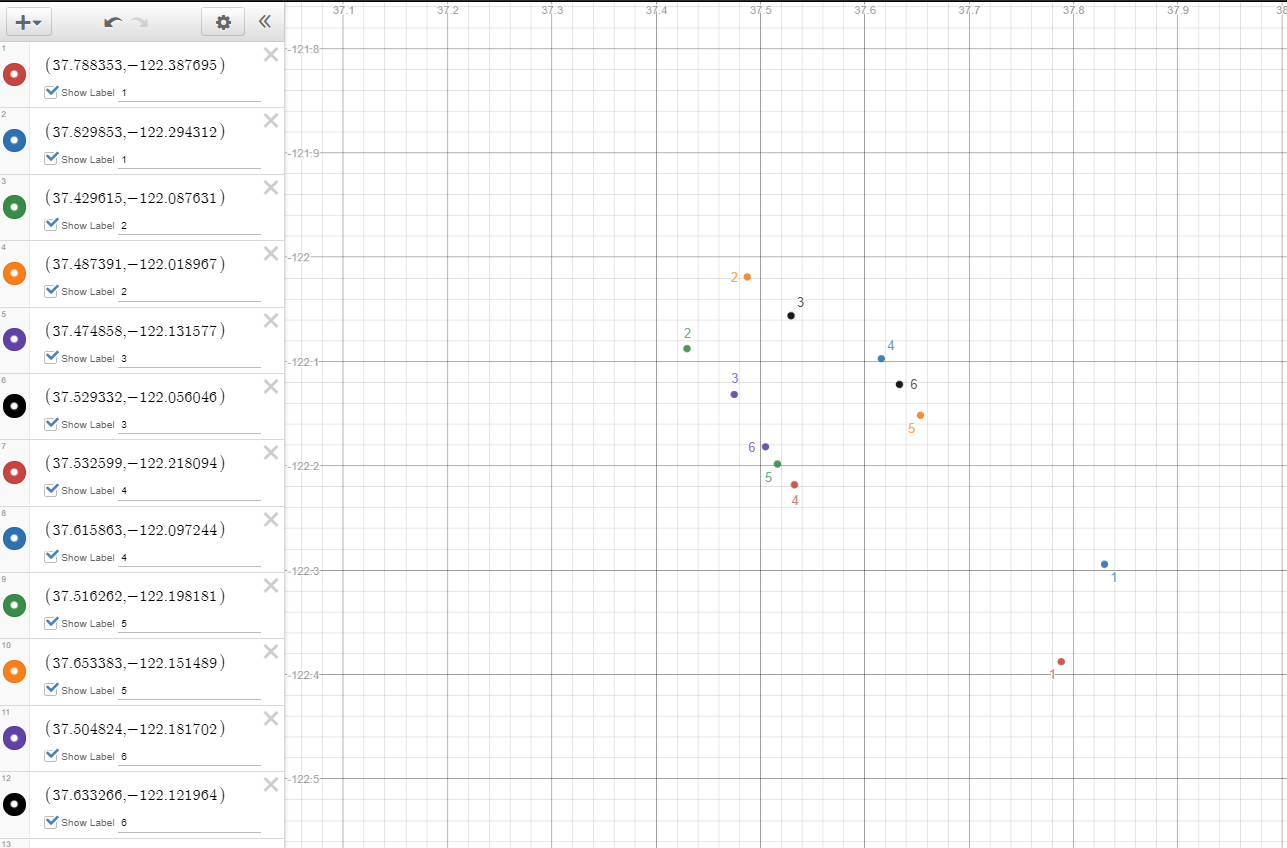
Our program will take in a file as an argument. Inside that file, we will have coordinates that might look like the following:

C:\Users\joopj\AppData\Local\Microsoft\Windows\INetCache\Content.Word\li.png

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:

C:\Users\joopj\AppData\Local\Microsoft\Windows\INetCache\Content.Word\li.png

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:

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 finite 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), x1y1 (representing 1 coordinate), and x2y2.

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.

We need a function that can go into a file to read, parse, extract information, and store information into a list. This function should do all that and then store that information into List 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.

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.

**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]

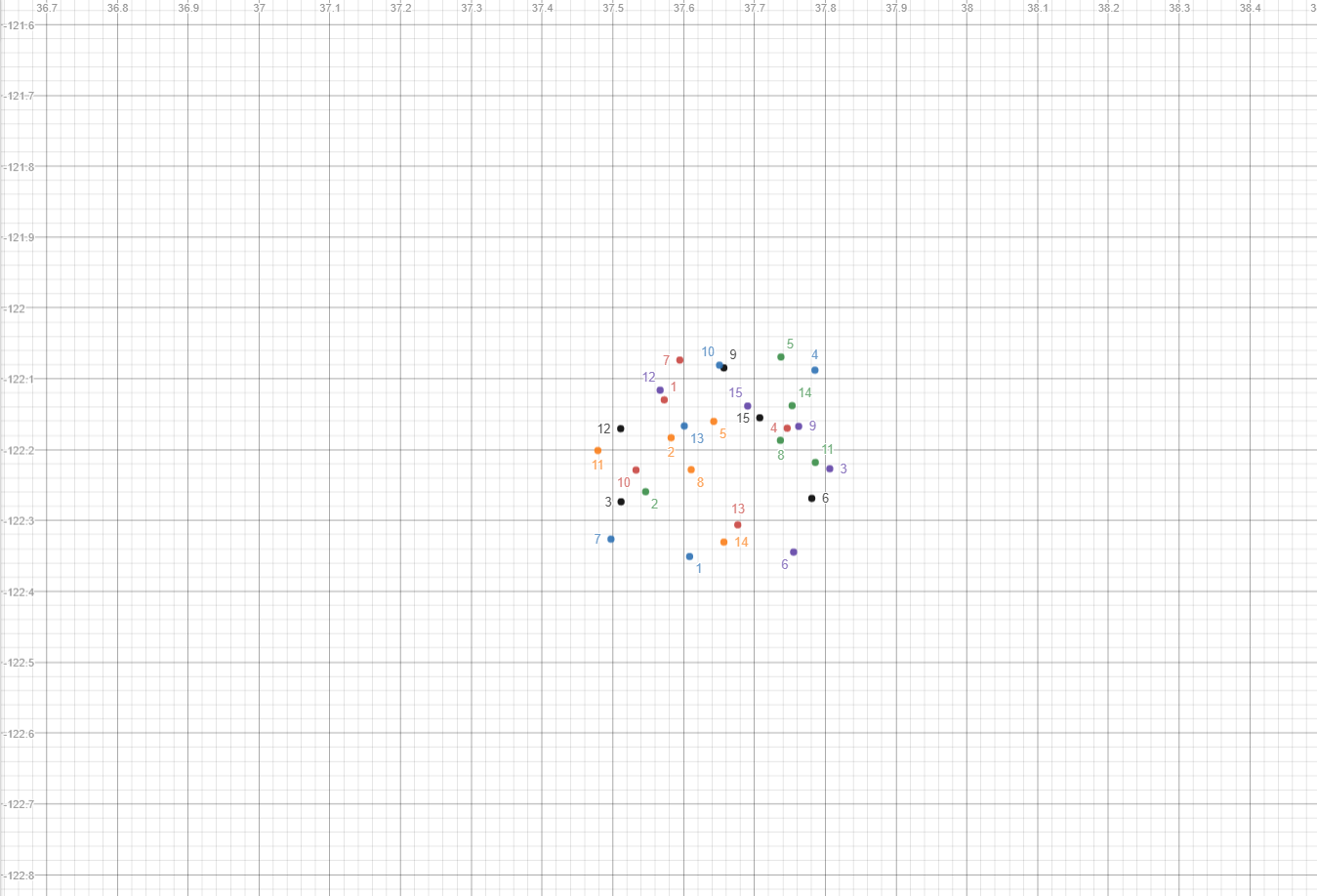
**Sample Output 1 (GIVEN):**

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:**

****

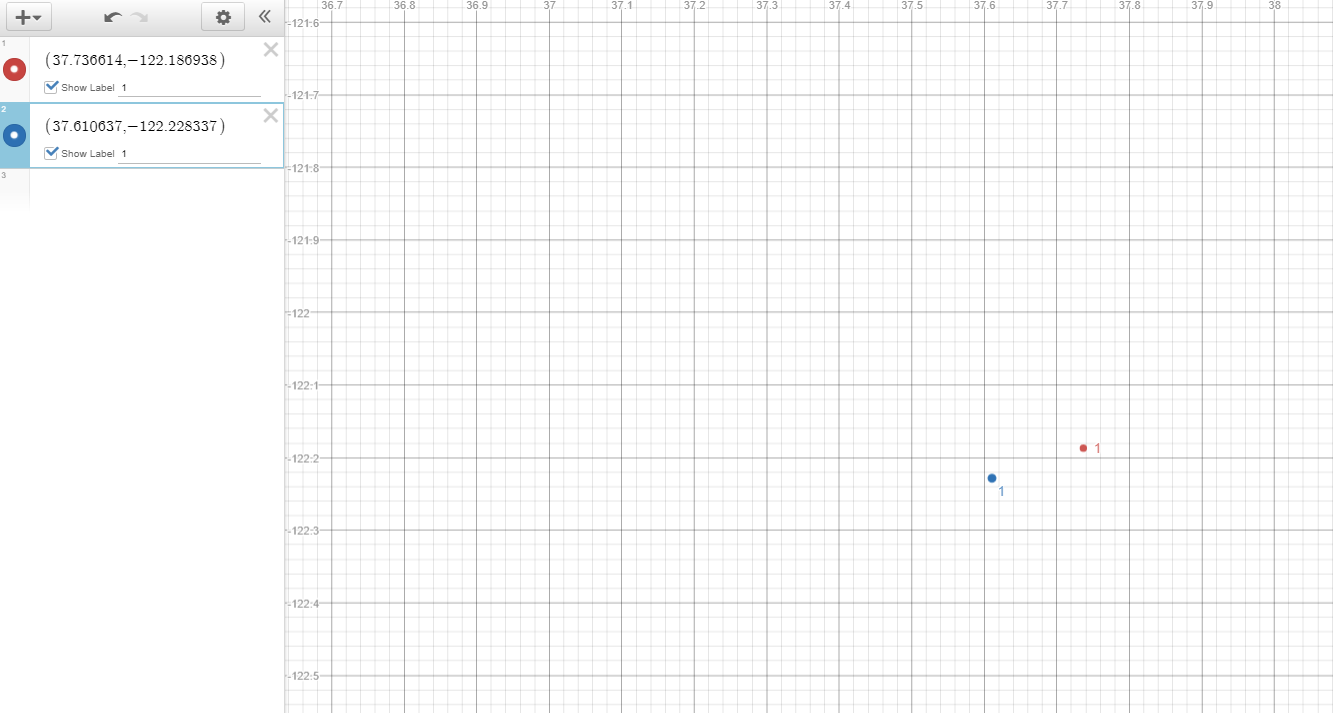
**Sample Output 2:**

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:**

****

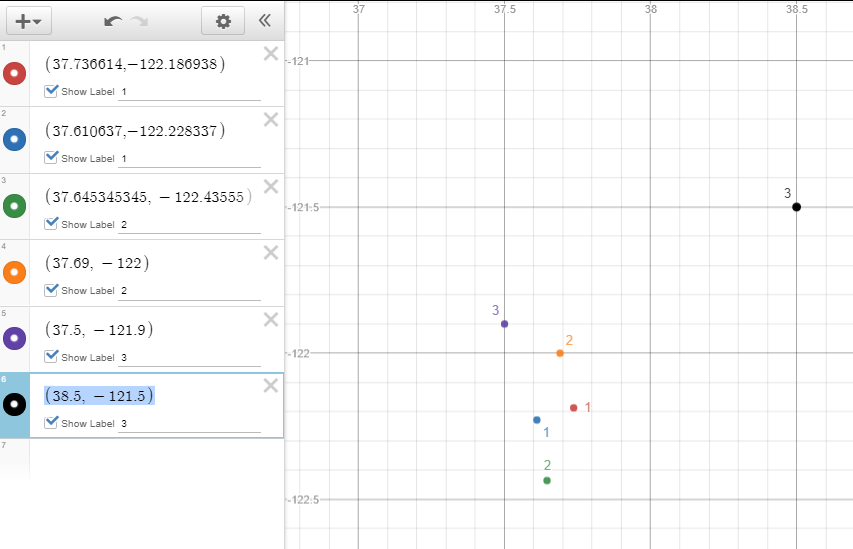
**Sample Output 3:**

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:**



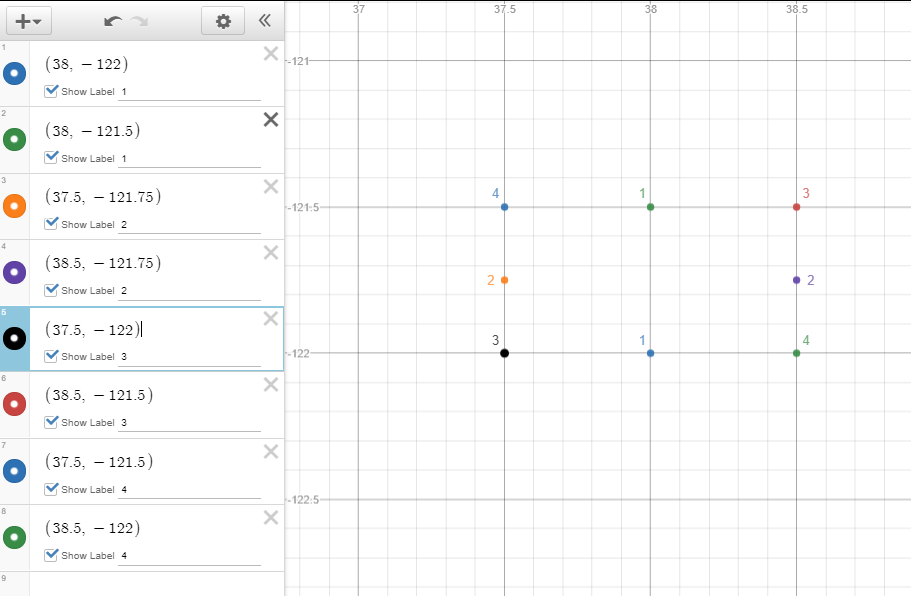
**Sample Output 4:**

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:**

****

**Sample Output 5:**

1

**Sample Input 6:**

[blank text file]

**Sample Output 6:**

Input file is blank.

**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.

1. 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 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.
2. Stack Overflow: What does if \_\_name\_\_ == “\_\_main\_\_”: do?  
   <https://stackoverflow.com/questions/419163/what-does-if-name-main-do>