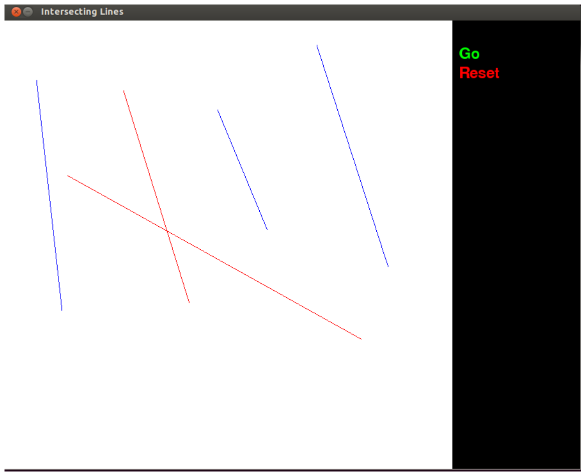
# C343 Project 3 - Intersecting Lines and Search Trees

Due 11:59pm, September 20, 2013

## 1 Project Description

For the project, the user can draw arbitrary lines on the board, some of which might intersect with each other. Once the user presses the 'Go' button, the program should highlight in red the leftmost intersecting lines.



We have provided you with the driver code that contains three modes:

- interactive: the user is shown the board. They can draw multiple lines on the board, some of which might intersect. All the drawn lines are colored with blue. The user then presses the 'Go' button, after which the leftmost intersecting lines will be colored red. To invoke this mode, run 'python driver.py'.
- gentests: here one can generate tests on a number of board sizes. The test files are stored in the tests directory. To invoke this mode, run 'python driver.py gentests'.
- test: in this mode, your code will be tested using the tests we generated. To invoke this mode, run 'python driver.py test'.

There are other supporting files, `segment_intersection.py`, `merge.py`, `stack.py` which help the driver run your code.

## 2 Task 1: Unbalanced Search Trees

You have to modify `BST.py`. There are two classes in that file, **BSTNode** and **BinarySearchTree**. The constructor for **BinarySearchTree** takes a function `less` as an argument. This method is used to compare keys in the tree. This is a way to make the class work with key types that support comparison like integers, floats, pairs of int or floats and so on. There is also a `root` argument which should be an instance of **BSTNode**. The `root` denotes the root of the tree. A **BSTNode** instance contains three fields:

- key - also known as value to be inserted in the tree
- left - left child of this node (another **BSTNode** instance) if present, `None` if there is no left child
- right - right child of this node (another **BSTNode** instance) if present, `None` if there is no right child
- parent - parent of this node or `None` if this node is the root.

Your task is to implement the following methods for **BinarySearchTree**:

- `insert(k)` - inserts a key (aka value) in the tree by creating a node for that key and it returns the new node. The `less` comparator provided in the constructor is used to determine the position of the new node in the tree.
- `successor(n)` - returns the successor of `n` in the tree
- `predecessor(n)` - returns the predecessor of `n` in the tree
- `search(k)` - returns node with value `k` if found else `None`.
- `delete_node(n)` - deletes the node `n` from the tree if present.

## 3 Task 2: Balanced Search Trees

You have to modify `avl.py`. Implement the search tree interface (same as for BinarySearchTree), but this time keep your tree balanced using the AVL approach.

## 4 Deliverables

Your github `project3` folder should contain all the files from the zip. These are the ones you need to modify:

- `BST.py` - your implementation of a binary search tree
- `avl.py` - your implementation of a AVL search tree

- A graph with two lines representing the execution times for running segment intersection in batch mode, comparing your BST and AVL implementations.
- `README.md` - where you explain your code

## 5 Driver

You can invoke the driver by running 'python driver.py'.