Cs345 — Programming assignment #1. Due July 13
Detecting Intersections between Segments. Submissions in pairs is allowed

1 Introduction

The input is a set $S = \{s_1 \ldots s_n\}$ of segments. Your program should print 'YES' if there is a couple of them that cross each other. Otherwise your program should print 'NO'. The running time of your program **must not** be more than $O(n \log n)$, where n is the number of segments.

Your grade in this assignment would be determined mostly for the data structures correct implementation and usages in the line-sweep algorithm.

Highlights:

1. You could must run in $O(n \log n)$ time. You have to implement the line sweep algorithm.
2. The Events queue (defined below) must be implemented with a binary heap.
3. The Status (defined below) must be implemented with a SkipList
4. You have to implement yourself both the binary heap, and the SkipList.
5. Sharing these parts with other students, or obtaining them from other sources is considered a cheating. Sharing code that deals with the graphics, input/output and users interface is allowed and welcome.

The operation of *highlighting* a point is drawing it with a different of color and/or size

So a segment connecting the point p_1 whose x- and $y-$ coordinates are (x_1, y_1) to the point p_2 whose coordinates are (x_2, y_2) will appear as a line in the input file,

$$x_1 \; y_1 \; x_2 \; y_2$$

The first segment is $\bar{s}_1$, the second is $\bar{s}_2$ etc until the last one which is $\bar{s}_n$. You program should print 'YES' if and only if there is a pair $\bar{s}_i, bsi$ that cross each other. That is, there is a point share by these two segments.

Once your program finds any pair of segments that intersects, in should print their numbers. For example "Segment s_3 crosses s_{17} "

1.1 GUI

Warning When using the default setting of Processing, the top line of the screen has $y-$ coordinate 0, the line below it has y coordinate 1 etc. That is, the above-below relationship seems revered.

The algorithm needs to display on the screen

1. The segments of the input set S, which are within the screen coordinates $0..800.$

   After reading the input file, your programs needs to initialize the data structures, and place the sweeping line on the left-most endpoint.
2. The sweeping line when within these coordinates in its currant location.
3. Highlight each pair of segments once you check whether they intersect.

After handling an event, the program will wait for a mouse click. Once the mouse is clicked, the program will proceed to handle the next event.

Your should use buttons (from ControlP5) or read user button click, to trigger the follow functions:

1. Ask for the name an input file, and after reading this file, initialized the heap
2. Clicking 'next' will move the line sweep to the next event, and will highlight which pairs of segments are being checked for intersections.

1.2 Which functions will you be given

Note: We are currently working on more starter code(To make the project more digestible!), will be released no later than 6/22/19

1. **Is_Above**$(s : segment, p : point)$ returns one of the four values {1,0,-1, ERROR}. If p is vertically above s it will return 1. If p is on s, it will return 0. If p is vertically below s the function will return -1. If p is none of these cases is satisfied, the it returns an ERROR. The latter case is shown in Fig 1.

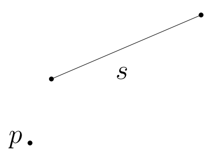


Figure 1: In the case that p is to the left of the left endpoint of s, or right of the right endpoint of s the function Is_Above will return ERROR

2. **Are_Intersecting**$(s1 : segment, s2 : segments)$ Returns TRUE if $s1$ crosses $s2$.

You will also be given classes "Segment" and "Point" which will be used as the primary data structure to represent segments and to use in the events.

Additionally, you will be given gui code to represent the segments and points. This will use the methods given in the "Segment" and "Point" classes.

Data structures and algorithmic comments

The status T must be stored as a SkipList. As discussed in the lectures. T contains the segments that cross the sweeping line ℓ, sorted from bottom to top. Note that the keys to be stored are not exact numbers, but rather names of segments. However, their order in the Skiplist is the sorted according to the order along the sweeping line ℓ from bottom to top.

When an event happens (left endpoint of a segments s or right endpoint of s, the algorithm must:

1. If the event is the **left** endpoint p of s (**birth event**) the algorithm should insert s into the status, stored in the SL. Then the algorithm finds the segments just above and just below s, and check if any of them intersects s. Finding the segment just above s is performed via $succ(p, T)$, the successor operation, when T is the status, and the segment below will be found using $pred(p, T)$, the predecessor operation.
2. If the event is the **right** endpoint p of s (**death event**) the algorithm should delete s from the Skiplist, find the segments s' and s'' just above and below p (if exist) and check if they intersect.

   algorithm must insert (resp. delete) s_i into (from) T, and check:

The running time must not exceed $O(n \log n)$

In order to sort the events, you could use any $O(n \log n)$ algorithm you prefer. Some might find it easier to implement the sorting in their own hands, while other could prefer to use other students' code, a library, or Google. All these options are legal and will not affect the final grade, but you must indicate the source of the code.(Otherwise plagiarism detection might kick in)

After the sorting, the set of $2n$ endpoints of all segments of S should be stored in a static array named Q. It should be sorted by increasing x-value, and with each endpoint in Q you maintain the satellite information - what is the id of the segment that contains this endpoint.

Your program should also contain a global **static** array SEG counting the n input segments.

You are allowed to use friends' and colleagues' code for the GUI, and file input/output. You are not allowed to share snippets of code dealing with the actual calculations of values and data structure manipulations.

1.3 Required Classes/Methods

This project has a lot of freedom as far as implementation goes. However, there is a few classes and methods we require.

Classes:

- **SkipList**: This is the SkipList implementation, you will most likely need more than these methods, but these are the only strictly required
- ———-getNext(): Will return the next node
- ———-getPrev(): Will return the previous node
- ———-int fields: height, width
- ———-Node fields: head, tail
- **Node:** This is the Nodes that will be stored in the skiplist
- ———-Node fields: up, down, left, right. These are the node references, use these names for the references.
- **main**: This is the main file, where all the executing code should go **minHeap:** This is your **Event:** This is the event class, this will represent points for the line to stop on

  Remember to comment your code so I know your thought process on your implementation!

  Additionally you will have to submit a short readme with your project describing how to use your program, this can be as short as a paragraph.

2 More instructions

1. When you start the program, it should as for an input file name (it can have a default name if name is not specified). Your program then shows all the segments, or their portion in the $0..800 \times 0..400$. Window, and also show the sweeping line, when it is position on the leftmost segment endpoint. Every push on the button 'next' will move the sweeping line to the next endpoint, specify which type of event it is (birth or death event), and highlight every pair of segments once you check if they intersect.

   You could add more buttons and functionality, and creativity and coolness are appreciated.

2. Submit a zip file containing your project named: **ImplementationHW1_*YourName*.zip**

3. The expected running time of your algorithm must be $O(n \log n)$

4. Submission in pairs is encouraged. If you cannot work with any other students (either by your own choice, or due to geographic constraints) you are allowed to submit the program by yourself. In this case, you could assume that all input segments are horizontal (this simplify the code a fair bit). If you opt for this option, you are not allowed to have any collaboration with other students. [1]

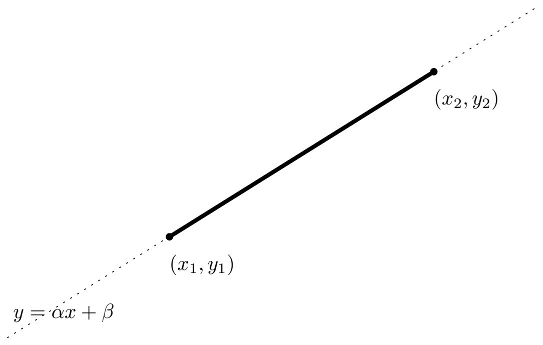
[1] Sorry that this sounds mean. I am trying to prevent a scenario when a couple of students that are working together might opt to submit each one his/hers copy, and earn the deduction of an easier input

Equation of a line that passes through two points p_1 and p_2

If we are given two points (x_1, y_2) and (x_2, y_2), the line that contains both of them can be written as

$$y = \alpha x + \beta$$

where $\alpha = \frac{y_2 - y_1}{x_2 - x_1}$ and $\beta = y_1 - \alpha x_1$.


(x_2, y_2)
(x_1, y_1)
$y = \alpha x + \beta$