COS 226 Programming Assignment Checklist: Point Location

Frequently Asked Questions

What should my output format be? Your output must have one line per query pair. To make things easier on the grader, precede each separated pair by `YES` and each non-separated pair by `NO`.

```
NO:  points (0.250, 0.800) and (0.600, 0.500) are not separable.
YES: points (0.030, 0.030) and (0.010, 0.220) are separated by line (0.000, 0.120) and (0.230, 1.000)
```

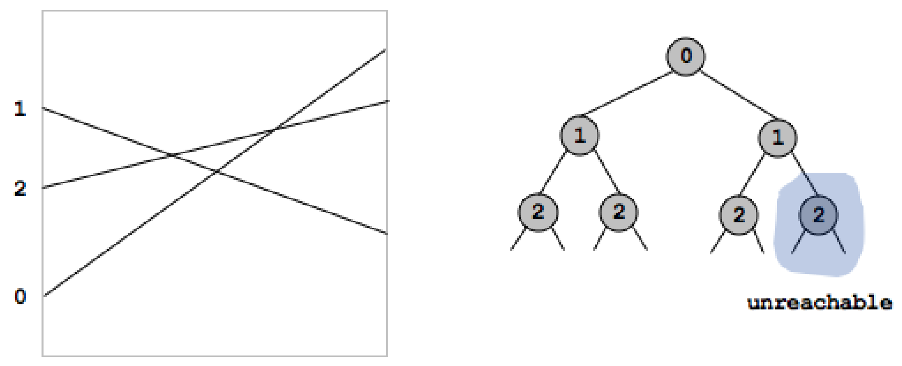
What should I do if many lines separate a query pair? Output any one such line.

Do I need to use the search tree approach described in the assignment? No. However, if you use another method, be sure to provide a good justification in your `readme.txt`.

If one or two of the points lie on an input line, does that line separates the two points? We'll leave that decision up to you. If you use floating point numbers, you won't be able to make such a fine distinction. For example, with `input5-99998yes.txt`, our reference solution claims that the points (0.725, 0.644) and (0.461, 0.816) are separated by the line between (1.000, 0.200) and (0.300, 1.000), even though the point (461/1000, 816/1000) falls exactly on the line given by y = -8/7 x + 47/35. That's ok as long as you document the cause in your readme file.

Should I get exactly the same number of nodes as the reference solutions? Your answer should be in the same ballpark, but don't expect an exact match. The difference is due the way you introduce roundoff error in your floating point calculations, and how you deal with degeneracy.

Why do I need to compute the line segment intersections? It's not strictly necessary for correctness, but if you don't sub-divide the intervals, you will compute alot of false intersections and blow up the size of your tree. The following input should generate a tree with only 7 external nodes (one corresponding to each subdivision of the unit square). If you detect that lines 1 and 2 intersect in the clockwise subtree, you will end up with four nodes containing line 2 (instead of three).



What is meant by an external node? A null link. Each external node should correspond to one (and only one) subdivision of the unit square.

What is meant by the external path length? See Sedgewick Definition 5.6 on page 237.

Should I compare every pair of line segments for an intersection when building the tree? No. You will need a separate node for each region, so it is possible that you will end up comparing roughly N^2 pairs of lines if there are that many intersections. However, your insertion algorithm should not take quadratic space unless there really are this many distinct regions. If you're not careful, you can end up with many nodes that represent regions of the plane and are unreachable via the ccw search.

How should I organize my program? The only requirement is that the `main` program is in a program named `PointLocator`. As always, use standard modular design principles. Our solution uses a data type `Point` for points in the plane, a data type `LineSegment` for directed line segments, and a search tree `PointLocator` for finding the region in which a query point lies.

Input, Output, and Testing

Input and output. Here are a number of sample input files. We also encourage you to create your own (possibly pathological) inputs to help test your program. We'll consider awarding extra credit if you submit inputs that cause other programs (or ours!) problems. The input should be very small, and it should expose a potential flaw that other programs are likely to face. We'll post these input files for other students to use.

Reference solutions. For reference, we provide executable code for our solution in Windows, Solaris, and OS X. Because of floating point and degenerate cases (and varying ways to deal with them), we cannot guarantee that our solutions are always accurate. If you discover any potential bugs, please let us know.

Timing. Unix users may find the shell script `timeit.csh` useful for computing a table of CPU times.

Submission and readme

You are free to organize your program as you see fit. As usual, we encourage and reward sound design principles. At the very least, you should probably have data types for points and line segments. Also, you should have a data type in which you can insert line segments and query for separations between point pairs.

Here is a template readme.txt file. It should contain the following information:

- A high-level description for the design of your algorithm, and what influenced your decisions.
- A table of the number of external nodes, and average path length of these nodes for various size input files. Discuss the growth of these numbers.
- A (brief) discussion of any vulnerability which your program has as far as degenerate cases or numerical inaccuracy.

Possible progress steps

- Download the directory `location`. It contains sample input files.
- Create a data type for points and line segments. Read in the input lines and point pairs and print them out to make sure everything is working.
- Given a (directed) line segment and a point, determine whether the point is on the counterclockwise or clockwise side of the line segment. Thoroughly test your routine.
- Given two (directed) line segments, determine if they intersect. Consider using your `ccw` test. Thoroughly test your routine.
- Given two line segments that intersect, return the point of intersection. Knowing the point of intersection itself is needed because you will want to break up a line segment at its intersection points as you insert it into the tree.
- Build the point location tree, but assume that no lines will intersect. Thoroughly test your program on an input file with no intersections.
- Modify your point location tree to handle the more general case when lines can intersect.

Enrichment

What's the most regions that can be created with N intersecting lines in the plane? (N^2 + N + 2) / 2. This is the famous pancake cutting problem.

Can you get logarithmic query times in the worst case? Yes. This was first discovered at Princeton by Dobkin and Lipton. They designed an algorithm that achieves O(log N) query time with O(N^2) space and preprocessing time. They also showed that if binary decisions are used to locate the query points, then you can't do better than log N time per query.

Quadratic space seems excessive. Can you do better? Yes, there are several more complicated algorithms that achieve the logarithmic query time in only linear space. The first such method was discovered at Princeton by Lipton and Tarjan. Chazelle (another Princeton professor) created something called a "hive graph" that also solves the problem in this bound. A simpler algorithm using persistent search trees discovered by Sarnak and Tarjan is the focus of a COS 423 lecture. The preprocessing step uses O(N) space and O(N log N) time; queries take O(log N) time.