CS325: Visible lines project

Prof. Borradaile

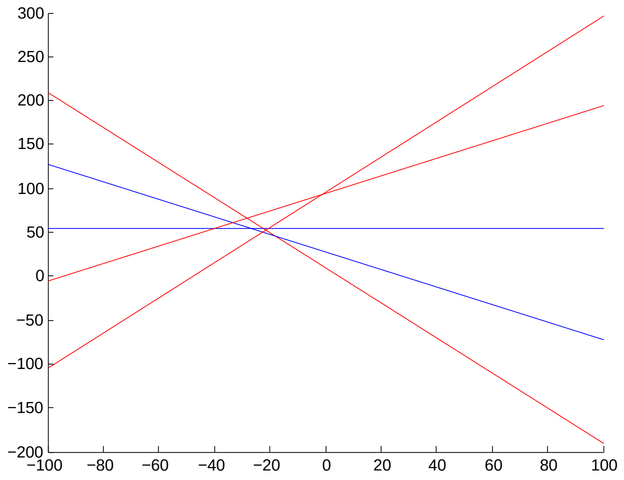
Due: Thursday, October 16 at 10AM

Your report must be typeset, printed and stapled*. Each team member's name must be listed as well as any resources used to finish the project. For questions regarding this project, please contact your TA, Spencer Hubbard at* `hubbarsp@eecs.oregonstate.edu`.

For this project, you will design, implement and analyze (both experimentally and mathematically) *three* algorithms for the visible line problem:

> Given lines $y_1, \ldots, y_n$ where $y_i(x) = m_i x + b_i$ and $m_1 < \ldots < m_n$, find the subset of visible lines. A line y_i is visible if there exists x such that for all indices j, $y_i(x) \geq y_j(x)$.

For example, if $y_1 = -2x + 9$, $y_2 = -x + 27$, $y_3 = 54$, $y_4 = x + 95$ and $y_5 = 2x + 96$, then the subset of visible lines is $\{y_1, y_4, y_5\}$. These are illustrated below; the red lines are the visible lines.



You may use any language you choose to implement your algorithms, but all three algorithms must be implemented in the same language. Plan ahead: in the next project you will implement a divide and conquer algorithm for this problem and you must use the same language for both projects – this may influence your choice of language. (You may use a different language for Projects 3 and 4 and the TSP Challenge.) Be sure that your algorithm is *correct* for any input.

Instructions

You will implement three algorithms for the visibility problem. The first two algorithms are based on the following claim, which we prove in class (Tuesday, October 7):

Claim 1: y_i is not visible if and only if then there exist j, k with $j < i < k$ such that $y_j(x_{j,k}) > y_i(x_{j,k})$ where $(x_{j,k}, y_j(x_{j,k}))$ is the point of intersection of the lines y_j and y_k.

Algorithm 1: Enumeration Initially, mark each line as visible. Loop over each triple of indices $j < i < k$ and compute the point of intersection $(x_{j,k}, y_j(x_{j,k}))$ of the lines y_j and y_k. If $y_j(x_{j,k}) > y_i(x_{j,k})$, then mark y_i as not visible.

Algorithm 2: Better Enumeration Notice that in the previous algorithm, the same line may be tested for visibility multiple times after it has been marked as not visible. Write a new version of the first algorithm that stops testing a line for visibility after it has been marked as not visible.

Algorithm 3: Even Better Enumeration The third algorithm is based on the following claim, which you will prove as part of this project:

> **Claim 2:** If $\{y_{j_1}, y_{j_2}, \ldots, y_{j_t}\}$ is the visible subset of $\{y_1, y_2, \ldots, y_{i-1}\}$ $(t \leq i-1)$ then $\{y_{j_1}, y_{j_2}, \ldots, y_{j_k}, y_i\}$ is the visible subset of $\{y_1, y_2, \ldots, y_i\}$ where y_{j_k} is the last line such that $y_{j_k}(x^*) > y_i(x^*)$ where $(x^*, y_{j_k}(x^*))$ is the point of intersection of the lines y_{j_k} and $y_{j_{k-1}}$.

This claim should allow you to iterate over indices i and find the visible subset of $\{y_{j_1}, y_{j_2}, \ldots, y_{j_k}, y_i\}$ given the visible subset of $\{y_1, y_2, \ldots, y_{i-1}\}$.

Testing for correctness Above all else, your algorithms should be correct. A file containing test sets will be found here (by Tuesday, October 7):`http://web.engr.oregonstate.edu/~glencora/cs325/visibility` The file has one test case per line. A line corresponding to the example above would be:

```
[-2, -1, 0, 1, 2], [9, 27, 54, 95, 96], [True, False, False, True, True]
```

You may use this test file to check that your code is correct. **Warning: Be sure to never use division! Doing so will result in unpredictable rounding errors that can result in incorrect solutions! See the note at the end of the accompanying notes.** (These notes will appear on Tuesday, October 7.)

Experimental analysis For the experimental analysis you will plot running times as a function of input size. Every programming language should provide access to a clock (not necessarily in seconds). Run each of your algorithms on input arrays of size $100, 200, 300, \ldots, 900$ and $1000, 2000, 3000, \ldots, 9000$ (that is, you should have 18 data points for each algorithm). The first two algorithms may be frustratingly slow, so you may compute running times for sizes $100, 200, 300, \ldots, 900$.

To do this, generate random instances using a random number generator as provided by your programming language. Remember to order your lines by increasing slope and that no two slopes should be the same. Note that you should not include the time to generate the instance.

Plot the running times for each algorithm in a single plot. Label your plot (axes, title, etc.). Include an additional plot of the running times on a log-log axis. See here for an explanation: `http://en.wikipedia.org/wiki/Log-log_graph` Note that if the slope of a line in a log-log plot is m, then the line is of the form $O(x^m)$ on a linear plot. You may also find these videos helpful: `http://www.khanacademy.org/math/algebra/logarithms/v/logarithmic-scale` and `https://www.khanacademy.org/math/probability/regression/regression-correlation/v/fitting-a-line-to-data`

For an example of an experimental analysis in java, see `http://algs4.cs.princeton.edu/14analysis/`

Project report

For each of the above algorithms, your report **must** include:

Run-time analysis Give pseudocode for each algorithm and an analysis of the asymptotic running-times of the algorithms.

Correctness Prove that Claim 2 is correct and, given this claim, prove that your design for Algorithm 3 is correct.

Experimental correctness To illustrate that your code is correct, determine the solution *and* value for each instance in the correctness file, which will be here (by Tuesday, October 7):`http://web.engr.oregonstate.edu/~glencora/cs325/visibility` Each line of this file is a different input. A line corresponding to the example above would be:

```
[-2, -1, 0, 1, 2], [9, 27, 54, 95, 96]
```

Experimental analysis Perform an experimental analysis and include plots as described above. *Note: Keep the data used for these plots; you will use them in the next project.*

Extrapolation and interpretation Use the data from the experimental analysis to answer the following questions:

1. For each algorithm, what is the size of the biggest instance that you could solve with your algorithm in one hour?
2. Determine the slope of the lines in your log-log plot and from these slopes infer the experimental running time for each algorithm. Discuss any discrepancies between the experimental and asymptotic running times.

Code Upload your code to T.E.A.C.H. Only one student from each group should do this.