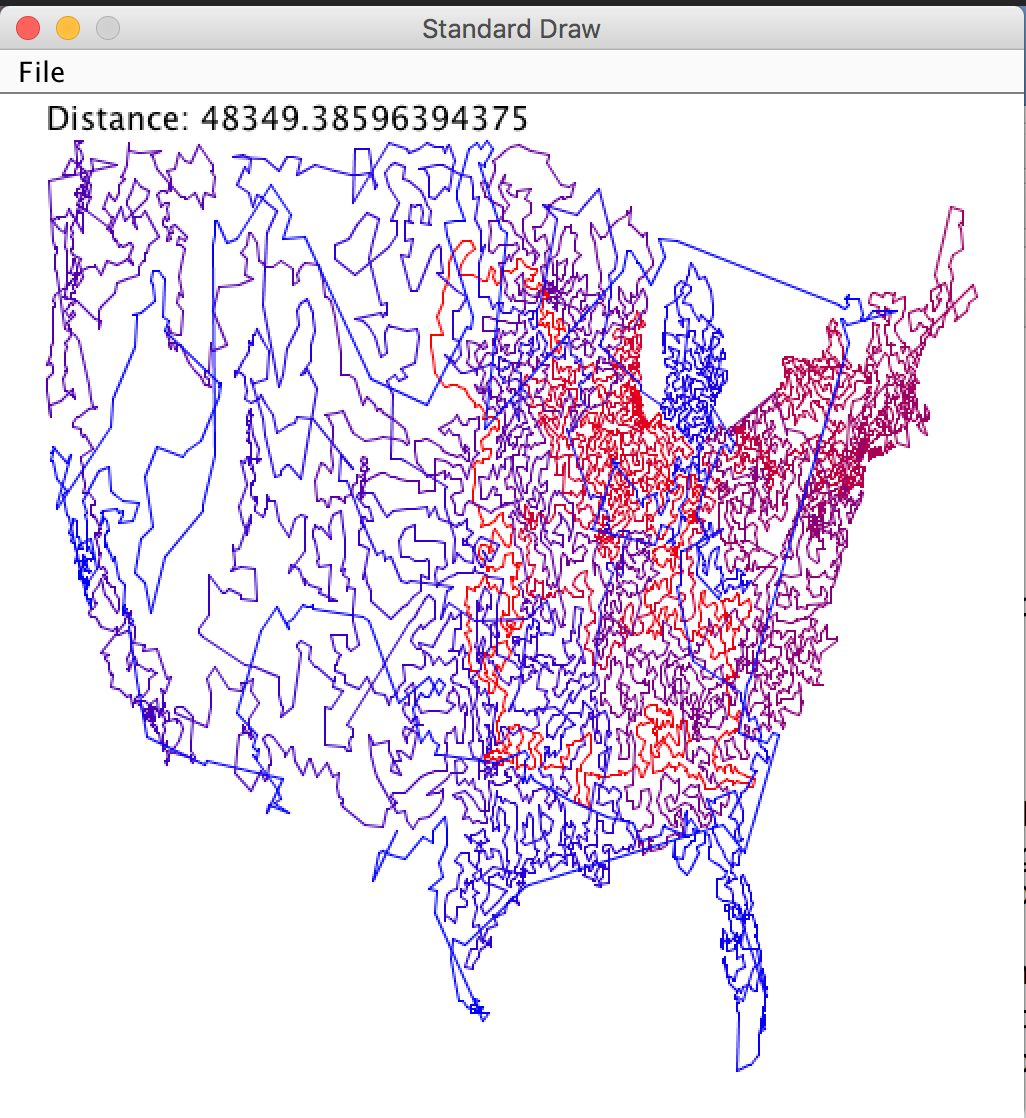
**Greedy Path**

Given a file containing a list of (two-dimensional) points, develop a program that does the following:

* Computes the "bounding box" of the set of points. That is, find the lower-left coordinate (x1, y1) and upper-right coordinate (x2, y2) of a rectangle that contains all points in the file.
* Construct a path visiting each point exactly once. The path always starts from the first point in the file. You construct the path in a ***greedy***fashion, always going next to the closest unvisited point.
* Compute the total distance of the greedy path you found through the points.

The lab folder contains some example input files, as well as a program you can use to graphically display your solution. Here is one of the provided input files, points7.txt:

7

0.0 1.0

1.0 1.0

-1.0 1.0

2.0 0.0

-2.0 -2.0

1.0 -2.0

2.0 -2.0

The first line in the file contains a number n, the number of points in the path. The next n lines are the pairs of points (space-separated X- and Y-coordinate) that make up the "path"

1. Begin by completing the class **Point.java**. Essentially an abstraction of a two-dimensional location (wrapping X- and Y-coordinates into one object), a Point can do the following:
   * Construct a new Point
   * Store and return its X- and Y-coordinate
   * Store and return if it has been visited (when traversing a path, i.e. a series of points)
   * Return the Euclidean distance (see the **FAQ** for info) between this Point and another Point supplied as a parameter
   * Return a String representation of this Point (an overridden toString() method)

All instance variables should be private; you should provide public getters and setters when necessary.

1. Complete the class **Path.java**. The Path type is simply a series of points, along with some statistics about the path. A Path can do the following:
   * Store the sequence and number of points that define it
   * Construct a new Path using the supplied String (representing a particular file)
   * Store and return the minimum and maximum X- and Y-coordinates in the path (used by the GUI to set the scale of the drawing)
   * Return the ith Point on the path (the Point at a particular index location)
   * Return the total distance traveled, going from Point to Point (in file order)
   * Return a String representation of this Path (an overridden toString() method)

Here are the sequential (non-greedy) distances for some of the provided files:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **points7.txt**  14.63441361516796 | **mona-20k.txt**  4964864.675750689 | **usa13509.txt**  3910558.8361292267 | |  |  |

1. Create a class **GreedyPath.java** that extends Path. GreedyPath should add the following:
   * The points (and their order) of a path using the greedy algorithm described previously
     + GreedyPath should NOT re-declare Path's array points (doing so will "hide" the super-class variable). GreedyPath should declare its own array, and fill it with values from the super-class' array (initialized in the super-class' constructor)
   * An overridden getDistance() method that returns the total distance traveled when traversing a path using greedy decision making
   * An overridden getPoint() method (returns the point at a particular index from the greedy path, not the original path)
   * An overridden toString() method

**We assume paths always start at the first point in the file.** **From there, the greedy path always goes to the *next closest point* that has not already been visited.** For example, in points7.txt, starting at (0.0, 1.0), the next closest point is (1.0, 1.0). From (1.0, 1.0), the next closest point would be (0.0, 1.0), but since we already visited this one, the next best one is (2.0, 0.0), and so on.   
  
You need to figure out which point is closest, given your current location. In the event of a tie, you should use the point that occurs first in the file. You will also need to keep track of whether each point has been visited or not (use Point objects' visited field).

GreedyPath should first utilize a super() call to initialize the "original" path. Next, write a private helper method void findPath() that will find and store the points that comprise the "greedy path" (the points in greedy order), as well as calculate and store the total distance traveled traversing the path in this fashion.  
  
You can assume the input file will be well-formed and contain at least one point. Here are the greedy distances for some of the provided files:

|  |  |  |
| --- | --- | --- |
| **points7.txt**  11.576491222541476 | **mona-20k.txt**  60618.72690198878 | **usa13509.txt**  48349.38596394375 |
|  | | |  |  |  |

If you are confident that your classes are working properly, use the provided program **DrawPath.java** to visualize the paths. DrawPath uses a GreedyPath instance that loads information from a text file. Examples:

|  |  |
| --- | --- |
| https://katie.mtech.edu/classes/csci135-online/assign/input/points7.png  **points7.txt** | https://katie.mtech.edu/classes/csci135-online/assign/input/mona-20k.png  **mona-20k.txt** |

**(Advanced) Shorter-est path**

Come up with an algorithm that provides a shorter path than the simple greedy heuristic described in the lab. You can use extra storage, more loops, go nuts!

*Adapted from the* ***Greedy Path*** *assignment*

[*katie.mtech.edu/classes/csci135-online/assign/input/*](https://katie.mtech.edu/classes/csci135-online/assign/input/)