CS 430 Programming Languages
Spring 2017
Module 12 PA

JAMES MADISON UNIVERSITY

Introduction

In this PA you will take the code you have written for manipulating Points and use it write a program that reads in a bunch of points, finds their convex hull, and then writes a Scalable Vector Graphics (SVG) file that can be displayed in a browser to show a plot of the points and the convex hull.

Points Module

Your first job is to make your code from Module 11 into a module called Points. Export the following names from the Point module: Point, Pair, Edge, distance, closestPair, convexHullEdges, convexHullPoints. Your module will of course have to be in a file called Points.hs.

Main Program

Download the file mod12PATemplate.hs from Canvas. Rename it mod12PA.hs; this will be the second of your deliverable files. This file contains several functions that generate SVG. Your program can use these functions to produce its output. In particular,

- All your output must be written to stdout (using actions like putStrLn). You can redirect this output into a file on the command line. The file should be called something.html because even though we are generating SVG, the browser may not process it properly unless it thinks it is getting an HTML file.
- The first data written to the output must be an SVG header. Such a string is generated by svgHeader, which must be supplied with the minimum and maximum values of the x and y coordinates of all the points to be graphed. This is needed so that the graph can be scaled properly.
- The next data written to the output sets up the SVG line drawing section and is generated by svgLineSection. This function also needs the minimum and maximum x and y coordinates (so it can draw axes) and also the line thickness. Because the graph is scaled, I recommend that this thickness be the maximum of the width and height of the graph divided by 300.
- The next data written to the output must be the line segments in the convex hull. The function edgeToSVG takes an Edge and returns a string with the appropriate SVG for drawing that Edge.
- Once all the SVG for drawing lines is written to the output, the next data written closes the line drawing section and opens the section for drawing points. The function svgPointSection generates the string needed for this job.
- Next comes SVG to draw all the points, which are small filled circles. The pointToSVG function takes a Point and the radius of the drawn circle and generates an SVG string for drawing a circle for that Point. The radius can be same as the line thickness, or a little bigger.

- The last data written to the output is the SVG footer. This string is generated by `svgFooter`.

The functions above are provided for your use in generating output. You must supply the remaining function and actions. The function and actions you must implement are specified as follows.

> `toPoints :: String -> [Point Float]`. This function must take a string containing pairs of floating point numbers. The pairs are separated by whitespace. Each pair consists of two numbers between parentheses separated by a comma. In other words, this is a string listing of Haskell values of type `(Float,Float)` or `Point Float`. This function must convert this string to a list of `Point Float` values. The idea is that the main program will read the input stream into a string, and this string must then be converted to a list of `Point Float` values. This function does the job.
>
> `printList :: [String] -> IO ()`. This action must take a list of strings and write each one to stdout, one per line.
>
> `main :: IO ()`. This is the main function. It must read `Point` values from stdin, separated by whitespace, until the end of the input stream. It must then write SVG to stdout that is a complete specification for a diagram displaying the x and y axes, the points read in, and the convex hull around the points. The lines and points must be in a color that is easily discernable agains the background (which must be white) by a person with normal color vision. The lines and points must be large enough to be easily discernible by a person with normal vision, but not so large that points obscure each other and lines ressemble rectangles. (I know these are vague requirements; ask if you are in doubt). You may use the colors currently built into the program (black for lines, red for points) or change them if you like.

The intent is that you can make a file containing points (for example, a file `points.txt` with contents (1,3) (-2, 8) (9,-3) etc. spread across lines arbitrarily), then you can run the program on the points file and write an HTML file (for example, `runhaskell mod12PA.hs <points.txt > tmp.html`). Finally, the HTML file can be opened in a browser, which will display the diagram.

Although you can write the main program with the supplied functions and actions plus those you are required to implement, if you feel the need to write additional helper functions or actions, you may do so.

Deliverable Requirement

Your program must be named `mod12PA.hs` and the `Points` module file `Points.hs`. These two files constitute the deliverables for this PA.

Submit your Haskell files in Canvas by the listed due date.