CS 430 Programming Languages
Spring 2017
Module 11 PA

Introduction

In this PA you will implement functions for manipulating points, pairs of points, and sets of points similar ot those you wrote in Ruby. I have prepared a template file called `mod11PATemplate.hs`. Please download this file and rename it `mod11PA.hs`. Then open the file and write your code in it, filling out the stubbed functions.

The Tests

If you look towards the bottom of `mod11PATemplate.hs` you will see a bunch of test functions. You should not change these functions in any way. If you load this file into `ghci` as is and run the `test` function, you will see that all the tests fail. The idea is that as you write your code, you can keep running `test` (or any of the other test functions) and determine whether your code is (probably) correct. I will also run `test` to see if you code is correct.

The Functions

Several of the functions have preconditions. If these are violated, the behavior of the program is undefined (in other words, it can do anything, including crash).

The template include definitions for `Point` (an ordered pair), `Pair`, and `Edge` (an ordered pair of `Points`). The functions you are to implement are specified as follows.

`distance :: (Real a, Floating b) => Point a -> Point a -> b`. This function must take two `Points` and return the distance between them.

`sqDistance :: (Real a, Floating b) => Point a -> Point a -> b`. This function must take two `Points` and return the square of the distance between them.

`pairsFromPoints :: Real a => [Point a] -> [Pair a]`. This function must take a list of `Points` and generate a list of all `Pairs` of points from the list. This results list should contain each `Pair` of points at most once. If the argument list contains fewer than two points, then the result list should be the empty list.

`closerPair :: Real a => Pair a -> Pair a -> Pair a`. This function must take two `Pairs` of points and return the `Pair` whose points are closer, or either `Pair` if there are equally far apart.

`closestPair :: Real a => [Point a] -> Pair a`. This function must take a list of `Points` and return the `Pair` from this list whose `Points` are the closest. The precondition of this function is that its argument contain at least two points.

`lineSide :: Real a => Edge a -> Point a -> a`. This function must take an `Edge` and a `Point` and return 1 (-1) if, as one moves along the edge from its first to its second `Point`, one must turn right (left) to reach the argument `Point`. This function must return 0 if the argument `Point` is on the line formed by the argument `Edge`. You may use the following formula to compute this function:
`lineSide ((e1x,e1y),(e2x,e2y)) (px,py) = signum ((px-e1x)*(e2y-e1y) - (e2x-e1x)*(py-e1y))`

isHullEdge :: Real a => [Point a] -> Edge a -> Bool. This function must take a list of Points and an Edge and return True just in case all the Points in the list are on the same side of, or on, the line formed by the Edge. In other words, it must return true just in case the Edge is part of the convex hull enclosing the list of Points. If the set of Points is empty, it must return True.

convexHullEdges :: Real a => [Point a] -> [Edge a]. This function must take a list of Points and returns a list of Edges forming the convex hull of the list of Points. The list of Edges must not contain any duplicate edges. If the set of Points contains fewer than two Points, the result must be the empty list.

convexHullPoints :: Real a => [Point a] -> [Point a]. This function must take a list of Points and return a list of the Points in the convex hull of the argument list. The result must not contain any duplicates. If the argument list contains fewer than two points the result must be the empty list.

Deliverable Requirement

Your program must be named mod11PA.hs. This file must have test code at the bottom unchanged from mod11PATemplate.hs.

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