CS 3110 Spring 2015
Problem Set 4
Version 1 (last modified March 20, 2017)

This is Part 1 of Problem Set 4

Please note that this is only part 1 of PS4. Part 2 will be released on Tuesday (3/21) along with a staff implementation of ordered fields of Rationals and Reals so you can test your solution of PS4 does not have to depend on PS3 and basic testcases.

Overview

This assignment focuses on computational geometry and features three algorithms: computing the area, convex hull, and Voronoi diagram of a set of points. The first two algorithms will extend your implementation of the real numbers in PS3 and show how modules and functors are powerful tools to extend an existing system. The third will test your overall proficiency in coding with a functional language.

If you consult any written or online sources you need to acknowledge them. Failure to do so is a violation of academic integrity.

You must write your own code. Code copied from outside sources with proper citation does not constitute an AI violation, but will receive minimal credit.

No imperative features are allowed throughout this problem set. This assignment must be done individually.

Objectives

- Be comfortable with using modules and functors to extend an existing system.
- Be familiar with concepts and algorithms in computational geometry.
- Be fluent enough in OCaml to be able to implement fairly involved algorithms in a functional style.

Recommended reading

The following supplementary materials may be helpful in completing this assignment:

- Lectures 11 12 13 14
- The CS 3110 style guide

- The OCaml tutorial
- Introduction to Objective Caml
- Real World OCaml Chapters 4, 6, 9

Compiling and testing your code

For this problem set we have provided a Makefile and minimal test cases.

- To compile your code, run `make main`
- To remove files from a previous build, run `make clean`

Please note that any submissions that do not compile will get an immediate 20% deduction from their original grade.

Revisiting fields

In this part of the problem set you will build upon your knowledge of the real numbers in PS3.

For the convex hull, you will need to use ordered fields that we have implemented from PS3. In part 2 of the problem set we will be releasing a staff implementation of the Fields so PS4 does not have to depend on PS3. However, in the meantime, please feel free to use your implementation if you want to test.

Exercise 1: Convex Hull.

In Euclidean space, a convex set is a region such that for every pair of points within the region, every point on the straight line segment that joins the pair is also within the region. Given a set of points X on the 2 dimensional plane the convex hull of the set is smallest convex set that contains X. There are various efficient algorithms to compute the convex hull of a set of points. In this exercise you will implement Graham scan which runs in $O(n \log n)$ where n is the number of points.

The algorithm can be described in 3 simple steps:

1. The first step is to find the point with the lowest y-coordinate. If the lowest y-coordinate exists in more than one point in the set, the point with the lowest x-coordinate out of the candidates should be chosen. Call this point $P = (x_P, y_P)$
2. Sort the set in increasing order of the angle they and the point P make with the x-axis.

   We introduce a function that maps a vector to a number in the interval $[-1, 3]$ which is monotonic with the angle the vector makes with the x-axis.

   Definition 1. *Given a vector (x, y). Let $p = \frac{y}{|x|+|y|}$. The pseudoangle it makes with the x-axis can be computed as:*

   $$p((x,y)) = \begin{cases} p & \text{if } x \geq 0 \\ 2 - p & \text{otherwise.} \end{cases}$$

3. The algorithm proceeds by considering each of the points in the sorted array in sequence. For each point, it is first determined whether traveling from the two points immediately preceding this point constitutes making a left turn or a right turn. If a right turn, the second-to-last point is not part of the convex hull, and lies inside it. The same determination is then made for the set of the latest point and the two points that immediately precede the point found to have been inside the hull, and is repeated until a "left turn" set is encountered, at which point the algorithm moves on to the next point in the set of points in the sorted array minus any points that were found to be inside the hull; there is no need to consider these points again.

Again, determining whether three points constitute a "left turn" or a "right turn" does not require computing the actual angle between the two line segments, and can actually be achieved with simple arithmetic only as follows.

For three points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ and $P_3 = (x_3, y_3)$, simply compute the z-coordinate of the cross product of the two vectors $\overrightarrow{P_1P_2}$ and $\overrightarrow{P_1P_3}$, which is given by the expression $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$. If the result is 0, the points are collinear; if it is positive, the three points constitute a "left turn" or counter-clockwise orientation, otherwise a "right turn" or clockwise orientation (for counter-clockwise numbered points).

TODO: : Implement the function `convex_hull : point list -> point list` in `geometry.ml` which returns the vertices of the convex hull that contains points in the input list in the counter-clockwise order.

TODO: : Write testcases for the `convex_hull` function.

We will provide some point sets using both modules `Q` and `R` in file `tests.ml` in part 2 of the problem set. Write test cases using these point sets and add a few more tests to make sure all cases are tested for. Recall that it is impossible to decide equality in $\mathbb{R}$, the set of real numbers. The `R.cmp` function provided will terminate if its inputs are distinct numbers but will loop forever otherwise. For this reason there are 2 cases `convex_hull` can fail on a point set.

- Points do not have distinct coordinates.
- There are 3 colinear points

In the real-coordinate point sets provided we ensure that these 2 cases do not happen. You need to keep this in mind while writing your test cases.

Exercise 2: Area of a polygon.

The convex hull in the previous exercise is an example of a convex polygon. In general a polygon is defined to be the area bounded by straight line segments which form a closed loop.

A polygon is self-crossing if one edge crosses over another edge of the same polygon. We will not consider this type of polygon.

Given the vertices of a non-self-crossing polygon, there is a simple algorithm that computes its area.

Let $[(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_{n-1}, y_{n-1})]$ be the vertices in **clockwise** direction.

Let $(x_n, y_n) = (x_0, y_0)$. The area A can be computed using the following formula:

$$A = \left(\sum_{i=1}^{n} (x_{i-1} + x_i).(y_{i-1} - y_i) \right) /2$$

If the list of vertices is in the counter-clockwise direction the result has the same magnitude as the area but with a negative sign in which case we can take the absolute value of A.

More explantion on the correctness of this algorithm can be found here.

TODO: Implement the function `area : point list -> t` in `geometry.ml` that computes the area of a polygon given a list of vertices in the clockwise or counter-clockwise direction. The return area must be positive.

TODO: : Write thorough testcases for the `area` function.

Comments

[written] At the end of the file, please include any comments you have about the problem set, or about your implementation. This would be a good place to list any problems with your submission that you weren't able to fix or to give us general feedback about the problem set.

Release files

The accompanying release file `ps4.zip` contains the following files:

- `writeup.pdf` is this file.
- `geometry.mli` and `geometry.ml` contain the interface and implementation of the solution to the first 2 geometry problems.
- `.ocamlinit` contain configuration options to include the `nums` package when running in `utop` without needing to specify it explicitly.