MCS 481 / David Dumas / Spring 2012

Project 3: Voronoi Diagrams

Due Friday, March 30

0. Overview

For this project you have two options:

- Coding option: Implement Voronoi decomposition.
- Experimental option: Study the Voronoi diagram as a point location structure with CGAL.

The options are described in detail below.

1. Coding option - Voronoi decomposition

Problem specification. Write a program that computes the Voronoi decomposition of the plane determined by three or more point sites that do not all lie on a single line.

Due to the relative complexity of efficient algorithms, full credit will be given for a submission that is either:

- Efficient — time complexity $O(n \log n)$ with linear storage, OR
- Robust — correctly handles degenerate input.

Thus, for example, it would be sufficient to implement Fortune's plane sweep algorithm in a way that handles non-degenerate input. In this setting "degenerate input" refers to any of the following:

- Two event points have the same x or y coordinate
- Three sites are collinear
- Four sites are concircular

The evaluation of your project will include testing your program on small sets of sites for verification of the output, and on large sets to confirm that your program can handle them.

Input. Read a list of sites from `stdin`. Each line will contain the coordinates x y of one site, separated by whitespace characters (e.g. space, tab). The list of sites may be followed by any number of lines containing only whitespace characters, which must be ignored.

Output. Write a description of the Voronoi diagram to `stdout`; use the following general form for the output:

> Vertex list
> ⟨blank line⟩
> Edge list
> ⟨end of file, or blank line⟩
> ⟨optional extra information⟩

Here the vertex list has one vertex per line, with coordinates separated by whitespace characters. The position of a vertex in this list determines its zero-based index (meaning 0=first vertex in list, 1=second, etc.), though this index is not part of the output.

Each line of the edge list contains a pair of zero-based indices of vertices that are connected by a line segment in the Voronoi diagram. It does not matter whether the list consists of

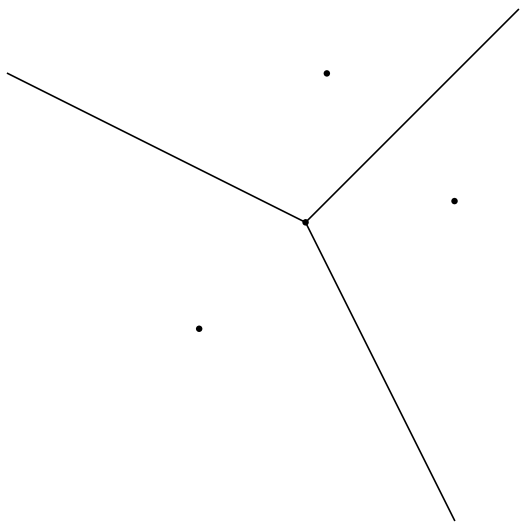

FIGURE 1. Voronoi diagram for the 3-site example.

oriented half-edges or unoriented edges, as long as each pair of vertices that are joined by a Voronoi edge appears in the list (in some order) and no other pairs appear.

The half-infinite edges require special handling. In order to represent them, we allow vertices *at infinity*. Unlike finite vertices, which are specified by a pair of coordinates (x, y), a vertex at infinity is specified by a single coordinate t in the interval $[0, 1)$. If an edge e has this vertex at infinity as an endpoint, then the edge is half-infinite and the clockwise angle from the positive x-axis to e is $2\pi t$. Thus, for example, the negative y-axis has endpoints $(0, 0)$ and 0.75.

Note that in the vertex list, the finite vertices can be distinguished from vertices at infinity because the latter will have only one coordinate.

Face information is not required in the output of your program, but it might be helpful for debugging. If the edge list is followed by a blank line then your program is allowed to write arbitrary additional information before exiting.

Examples. Note that the output of the program is not uniquely determined by the specifications above because a given Voronoi diagram could be described in many ways (e.g. re-ordering the vertex and edge lists). The following examples show conforming output for two Voronoi diagrams.

(1) Sample input for 3 sites:

```
0 1
1 0
-1 -1
```

Sample output for 3 sites:

```
-0.166667 -0.166667
0.125
0.426208
0.823792

0 1
0 2
0 3
```

There are three edges, each is half-infinite. The diagram is shown in Figure 1.

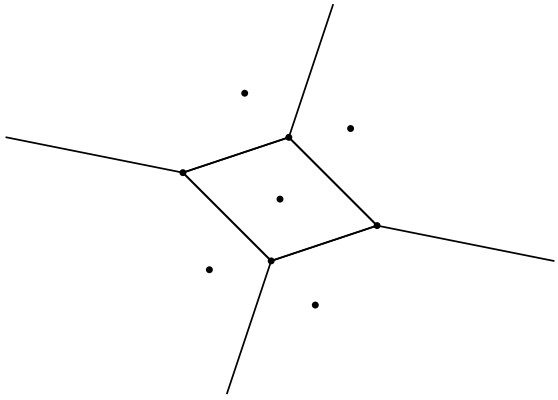

FIGURE 2. Voronoi diagram for the 5-site example.

(2) Sample input for 5 sites:

```
0 0
-1 3
1 -3
2 2
-2 -2
```

Sample output for 5 sites:

```
-2.75 0.75
-0.25 -1.75
0.25 1.75
2.75 -0.75
0.198792
0.468584
0.968584
0.698792

0 1
0 2
0 5
1 3
1 7
2 3
2 4
3 6
```

There are eight edges, four are half-infinite. There is one bounded cell corresponding to the site at the origin. Its vertices have indices $0, 1, 3, 2$. The diagram is shown in Figure 2.

Languages and libraries. The same programming language and library policies apply as in previous projects. Contact me for approval if necessary.

What to submit. Write a report on your implementation process. Start by briefly describing the problem and the algorithm. Then focus on the design decisions that were involved in your implementation.

Create a few small test cases for your code and verify the correctness of the output. Include at least one test case that has a bounded face (i.e. one with no half-infinite edges).

Submit the report, source code, and test cases by email (`ddumas@math.uic.edu`), following the same standards for report and source code submissions as the previous projects.

2. Experimental option - Voronoi diagrams and point location

Experiment 1: Voronoi decomposition. Study the running time of CGAL's functions for computing a Voronoi diagram. It will be necessary to generate lists of point sites, which can be accomplished using the random point generators from project 1 or some modification thereof. Attempt to create different types of site input that will result in more or less complex Voronoi faces, and see whether this is reflected in the running time of the algorithm.

In addition to collecting and analyzing timing data, answer the following discussion questions in your report:

- How does CGAL compute the Voronoi decomposition?
- Is this method related to the approach we discussed in class?
- What is the asymptotic complexity of CGAL's method?
- Were you able to observe different running time growth depending on the type of input? If so, can you explain the differences theoretically?

Also collect some statistics on the number of vertices and the number of bounded and unbounded edges of a typical Voronoi diagram (for some type of site input) and comment on the results. For example, if the sites are n random points in the unit disk, how does the typical number of unbounded edges grow as a function of n, and why?

For these experiments you can use the program `voronoi` included with the project description, which reads a list of sites from `stdin` and calls CGAL functions to compute the associated Voronoi decomposition. If invoked with an empty command line and standard or redirected input, e.g.

```
./voronoi
./voronoi < site-file.txt
```

then it will compute the Voronoi diagram and print some information about it (vertex and edge counts). Since there is some cost associated with the computation of the number of vertices and edges, for timing purposes it may be preferable to compute the Voronoi diagram and exit immediately with no output. To select this "quiet" behavior, use the command line option `-q`, e.g.

```
./voronoi -q
./voronoi -q < site-file.txt
```

Experiment 2: Point location and nearest neighbors. CGAL can perform point location queries on Voronoi diagrams. Since the face of the Voronoi diagram containing a query point q is the cell of a site p which is closest to q among all sites, this is equivalent to finding the *nearest neighbor* of the query point among the sites.

Study the running time of this type of nearest neighbor query with CGAL as a function of both the site configuration and the set of query points. An example program that enables this kind of testing is included with the project description and described below; your two main tasks in this experiment will be:

(1) Learn enough about CGAL's support for point location in Voronoi diagrams (using the online documentation) to develop hypotheses about the query time and its dependence on the site configuration.
(2) Design site configurations and timing experiments to test your hypotheses.

The program `nearest` reads a list of sites from a file and constructs the Voronoi diagram. Query points are then read from `stdin` and the nearest neighbor site for each one is determined (using CGAL's Voronoi point location function). With input redirection, site and query points can be read from files as follows:

```
./nearest site-file.txt < query-file.txt
./nearest -q site-file.txt < query-file.txt
```

By default the output is verbose, describing the face that contains the query point by listing its vertices in counterclockwise order. If the option `-q` is specified, a simpler output format is used: Each line contains the coordinates of the nearest neighbor vertex of a single query point.

Since it will be important to distinguish between construction and query time for this experiment, `nearest` measures these times separately and prints the results to `stderr`. Thus for example the following command will perform queries, write verbose point location information to `results.txt`, and record timing information in `times.txt`:

```
./nearest site.txt < query.txt > results.txt 2> times.txt
```

What to submit. Write a report that describes your hypotheses, experimental design, the data you collected, and your interpretation of the results. The report should have one section for each of the experiments described above. It is not necessary to include tables listing all of the raw data; rather, include graphs and tables that summarize enough of the data collected to justify your conclusions.

Submit the report by email (`ddumas@math.uic.edu`), following the same standards for experimental option and source code submissions from the previous projects. For the report, a single PDF file is preferred.