Voronoi Diagram

Final Assignment

Algorithms-I, Autumn 2008

Praveen Kr. Sharma
07EC1045
ornulu@gmail.com

Naveen Kr. Sharma
07CS3027
nkrsharma@gmail.com

November 12, 2008

Instructor: **Prof. Partha Bhowmick**

Evaluated By: **Mr. Rakesh Ranjan**

Department of Computer Science & Engineering

Indian Institute of Technology, Kharagpur

1 Problem Statement

1.1 Part 1 :

Given a binary matrix in which '1' denotes a point, prepare a set P of n sites such that the integer coordinates of each site are the row and the column indices of the corresponding 1 in the matrix. Use lexicographical sorting (vertically) and enqueue the sorted sites in the event queue. Label the sites with uppercase English letters. Repeat a label only if there are more than 26 sites. That labels of sorted sites is, if there are 28 sites, then their labels, ordered vertically down, are A (1st site), B (2nd site), C (3rd site), ..., Z (26th), A (27th), and B (28th site).

Input:

```
0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
```

Output:

```
0 0 0 0 0 A 0 0 0 0 0 B 0 0 0 C 0 0 0 0
0 0 D 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 E 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 F 0 0 0 0 0 0 0 0 0 0 0 0 G 0 0
0 0 0 0 0 0 0 0 0 0 H 0 0 0 0 0 0 0 0 0
```

1.2 Part 2:

Using Fortune's algorithm, construct the Voronoi diagram $V(P)$. Report $V(P)$ for a given position of the partial $V(P)$ when the sweep line meets F sweep line, as specified by the user. For each entry '0' in the input binary matrix lying above the beach line, mark it with the lowercase English character (e.g., 'b') corresponding to its nearest site (e.g., 'B'). If there is a tie (e.g., between 'B' and 'C'), then consider the site with smaller label ('b'). Mark the '0's through which the sweep line passes, by '-'.

Output:

```
d d d d a A a a e b b B b b c C c c c 0
d d D d a a a e e e b b b b c c c c 0 0
0 0 d 0 0 0 0 e E e 0 0 0 0 0 0 0 0 0 0
- - - - F - - - - - - - - - - - - G - -
0 0 0 0 0 0 0 0 0 0 H 0 0 0 0 0 0 0 0 0
```

For better Visual Appeal, we will be using Sun Microsystem's **Java Development Kit 1.6 Update 7** (`jdk-1.6.0_07`) for implementing this part of the problem.

2 Fortune's Algorithm

2.1 Voronoi Definition

Let $P = p_1, p_2, ..., p_n$ be a set of points in the plane (or in any dimensional space), which we call sites. Define $V(p_i)$, the Voronoi cell for p_i, to be the set of points q in the plane that are closer to p_i than to any other site. That is, the Voronoi cell for p_i is defined to be:

$$V(p_i) = \{q \mid dist(p_i, q) < dist(p_j, q), for\ j \neq i\}$$

The Voronoi diagram of a set of points is the planar subdivision whose faces are the voronoi cells of the points.

2.2 Introduction

Fortune's algorithm is a plane sweep algorithm for generating a Voronoi diagram from a set of points in a plane using $O(nlogn)$ time and $O(n)$ space. It was originally published by Steven Fortune in 1986 in his "A sweepline algorithm for Voronoi diagrams." It is the fastest algorithm for computing the Voronoi diagram for a set of $2D$ points. Fortune's algorithm simulates the growth of a *beach line* as the sweep line moves downward, and in doing so, traces the paths of the *breakpoints* as they travel along the edges of the Voronoi diagram.

2.3 Observations

Voronoi edges: Each point on an edge of the Voronoi diagram is equidistant from its two nearest neighbors p_i and p_j. Thus, there is a circle centered at such a point such that p_i and p_j lie on this circle, and no other site is interior to the circle.

Voronoi vertices: It follows that vertex at which three Voronoi cells $V(p_i)$, $V(p_j)$, and $V(p_k)$ intersect is equidistant from all sites. Thus it is the center of the circle passing through these sites, and this circle contains no other sites in its interior.

Degree: If we assume that no four points are cocircular, then the vertices of the Voronoi diagram all have degree three.

Convex Hull: A cell of the Voronoi diagram is unbounded if and only if the corresponding site lies on the convex hull. Moreover, each bounded cell of the Voronoi diagram is a convex polygon.

Size: If n denotes the number of sites, then the Voronoi diagram is a planar graph (if we imagine all the unbounded edges as going to a common vertex infinity) with exactly n faces. It follows from Euler's formula that the number of Voronoi vertices is at most $2n - 5$ and the number of edges is at most $3n - 6$.

2.4 Key Points and Major Steps

Sweep Line: The sweep line is a straight line, which we may by convention assume to be horizontal and moving downwards across the plane. At any time during the algorithm, the input points above the sweep line will have been incorporated into the Voronoi diagram, while the points below the sweep line will not have been considered yet. It also acts as the *directrix* for the construction of the parabolas in the beach line.

Beach Line: The beach line is not a line, but a complex curve above the sweep line, composed of pieces of parabolae. It divides the portion of the plane within which the Voronoi diagram can be known, regardless of what other points might be below the sweep line. For each point p above the sweep line, one can define a parabola (using the sweep line as the *directrix*. The beach line is the lower envelope (pointwise minimum) of these parabolae.

Site Events: When the sweep line passes over a new site a new arc will be inserted into the beach line. Let p_i be the current site. We shoot a vertical ray up to determine the arc that lies immediately above this point in the beach line. Let p_j be the corresponding site. We split this arc, replacing it with the triple of arcs p_j, p_i, p_j which we insert into the beach line. Also we create new (dangling) edge for the Voronoi diagram which lies on the bisector between p_i and p_j. Any old triple that involved p_j as its center arc is deleted from the priority queue, and we generate new events for each of the possible three new triples that result from this insertion.

Vertex Event: When the length of a parabolic arc shrinks to zero, the arc disappears and a new Voronoi vertex *may* be created at this point. In contrast to site events, vertex events are generated dynamically as the algorithm runs. Let p_i, p_j, and p_k be the three sites that generate this event (from left to right). We delete the arc for p_j from the beach line. We create a new vertex in the Voronoi diagram, and tie the edges for the bisectors (p_i, p_j), (p_j, p_k) to it, and start a new edge for the bisector (p_i, p_k) that starts growing down below. Finally, we delete any events that arose from triples involving this arc of p_j, and generate new events corresponding to consecutive triples involving p_i and p_k.

2.5 Data Structures Used

Height Balanced AVL Tree: The beach line is represented using an AVL Tree. An important fact of the construction is that we do not explicitly store the parabolic arcs. They are just their for the purposes of deriving the algorithm. Instead for each parabolic arc on the current beach line, we store the site that gives rise to this arc.

The operations that we will have to perform on the beach line are:

- Given a fixed location of the sweep line, determine the arc of the beach line that intersects a given vertical line. This can be done by a binary search on the breakpoints, which are computed using AVL Tree.
- Compute predecessors and successors on the beach line.
- Insert an new arc p_i within a given arc p_j, thus splitting the arc for p_j into two. This creates three arcs, p_j, p_i, and p_j.

- Delete an arc from the beach line.

A standard AVL Tree is easily modified to perform these operations in $O(logn)$ time each. We have created a `class AVLTree` to implement this data structure.

Event Queue: The event queue is a priority queue with the ability both to insert and delete new events. It is implemented as a binary heap, so all operations take $O(logn)$ time. Also the event with the smallest y coordinate can be extracted. For each site we store its y coordinate in the queue. For each consecutive triple p_i, p_j, p_k on the beach line, we compute the circumcircle of these points. If the lower endpoint of the circle (the minimum y coordinate on the circle) lies below the sweep line, then we create a vertex event whose y coordinate is the y coordinate of the bottom endpoint of the circumcircle. We store this in the priority queue. Each such event in the priority queue has a cross link back to the triple of sites that generated it, and each consecutive triple of sites has a cross link to the event that it generated in the priority queue.

The algorithm proceeds like any plane sweep algorithm. We extract an event, process it, and go on to the next event. Each event may result in a modification of the Voronoi diagram and the beach line, and may result in the creation or deletion of existing events. We have used the built-in `class PriorityQueue` to implement this data structure. It takes $O(logn)$ time to insert, delete and retrieve elements from the queue.

(Partial) Voronoi diagram: The partial Voronoi diagram that has been constructed so far will be stored in a DCEL. There is one technical difficulty caused by the fact that the diagram contains unbounded edges. To handle this we will assume that the entire diagram is to be stored within a large bounding box. We have created a `class DCEL` to store the finalised and dangling edges.

As there are $O(n)$ events to process (each being associated with some feature of the Voronoi diagram) and $O(logn)$ time to process an event (each consisting of a constant number of AVL Tree and Priority Queue operations) the total time is $O(nlogn)$.

Bugs & Issues: Our implementation of the Fortune's Algorithm fails to give the correct output in the following cases:

- When 4 concyclic points(sites) are present in the input. (This is a limitation of Fortune's Algorithm. It can be countered by shifting one of the points by a vey small amount).
- When points are very close together and more than one event is processed per pixel. In such a case, one pixel movement of the sweepline causes multiple events (site and circle events) which do not get processed properly. Hence the diagram generated is distorted.

In general, if points are evenly spaced apart on the canvas, our program generates a correct diagram.

3 Test Results

3.1 Execution Details

The zip file contains 3 source codes: `part1.c, Voronoi.java, gen_input.c`, 1 executable `jar` file, 5 sample input files, 5 output files (corresponding to part 1 of the assignment).

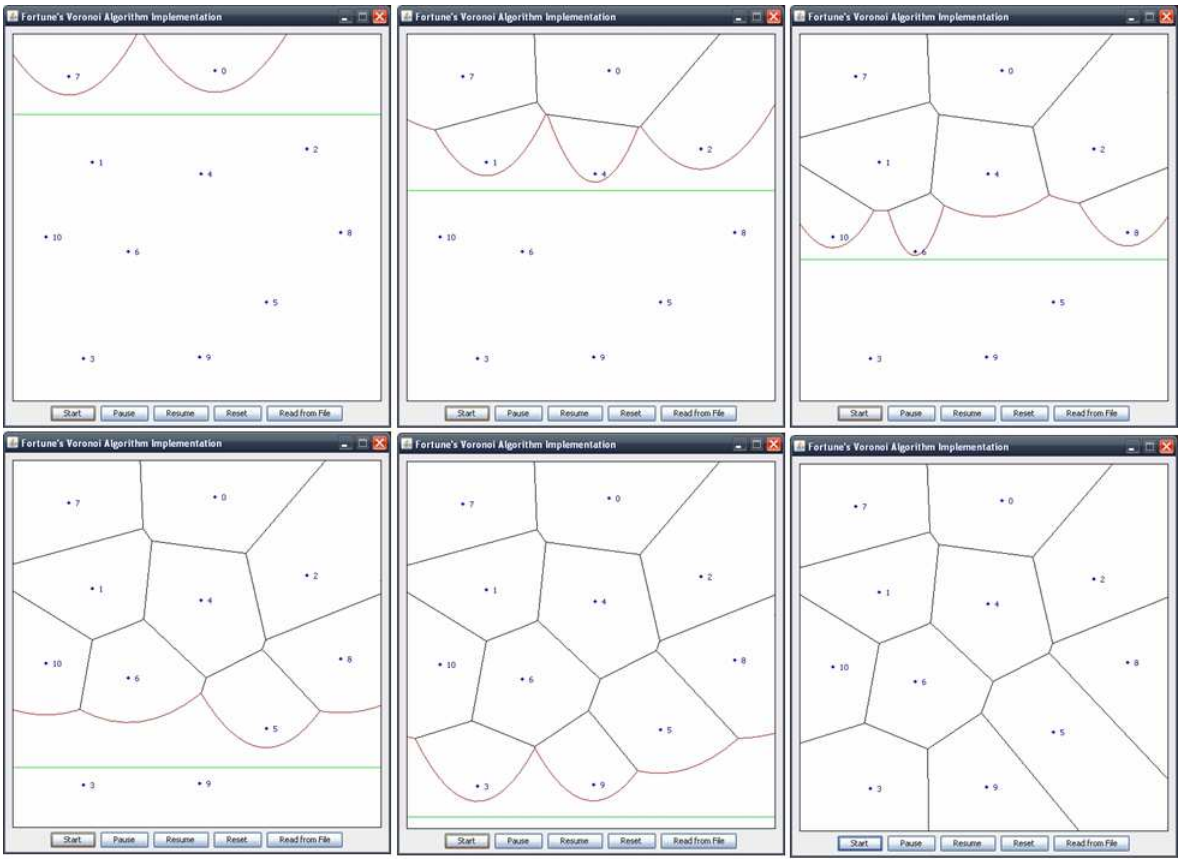
The format of the input files is as follows: The first line of the input file is number of rows and columns. The next `n` lines contain `n` integers each (either 1 or 0). 1 denotes a site in the voronoi diagram whereas 0 denotes blank space.

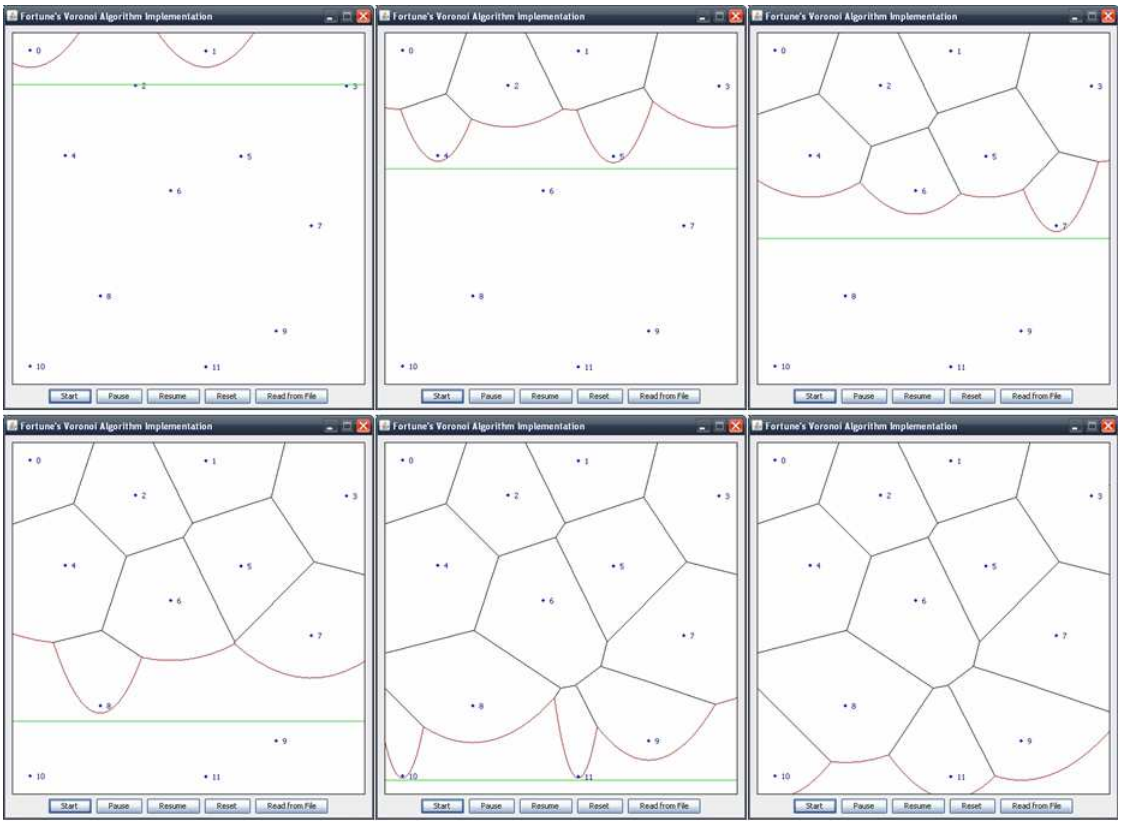
`part1.c` : This C program implements the first part of the assignment. It takes a filename as the input and writes the output to another file.

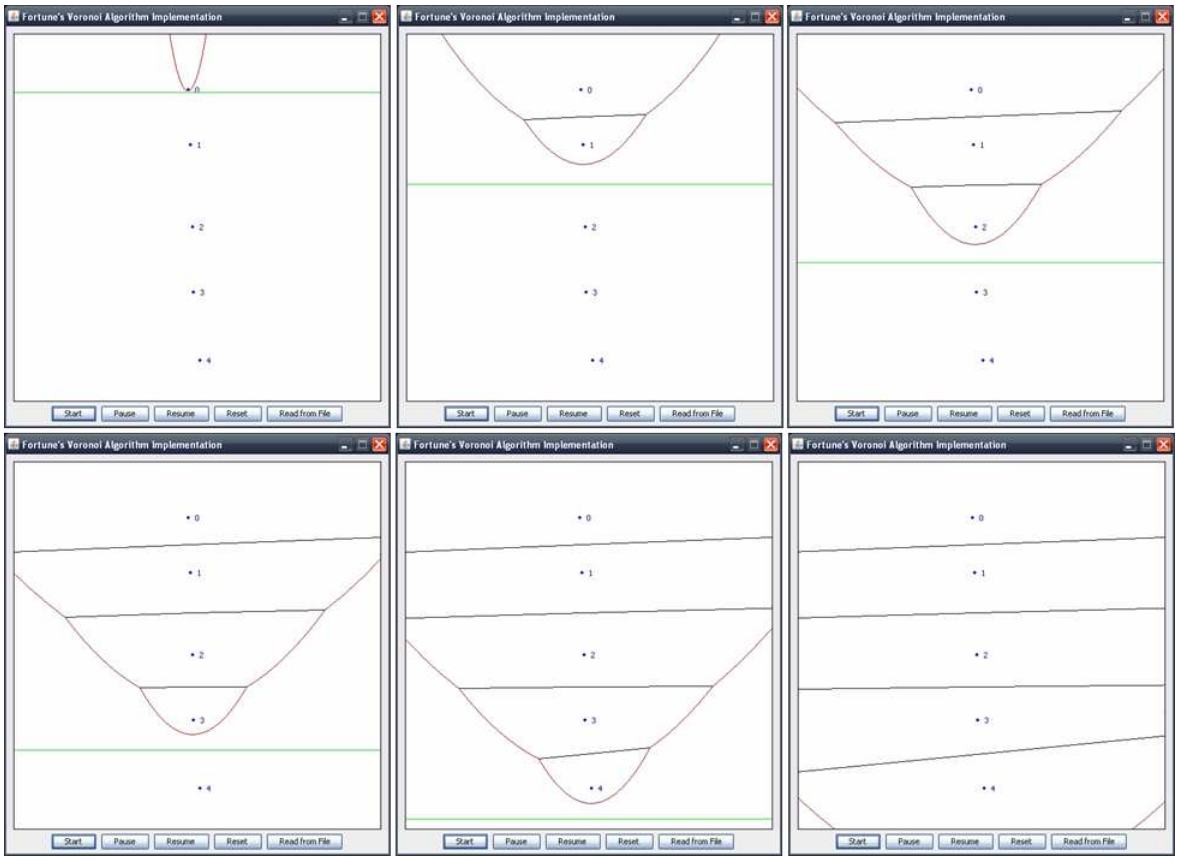
`Voronoi.java` : This JAVA program implements the 2nd part of the assignment. It can either take points from the user (via mouse clicks) or from any of the input files. The start, pause and resume can be used to simulate the growth of the beach line and view its structure at any instant.

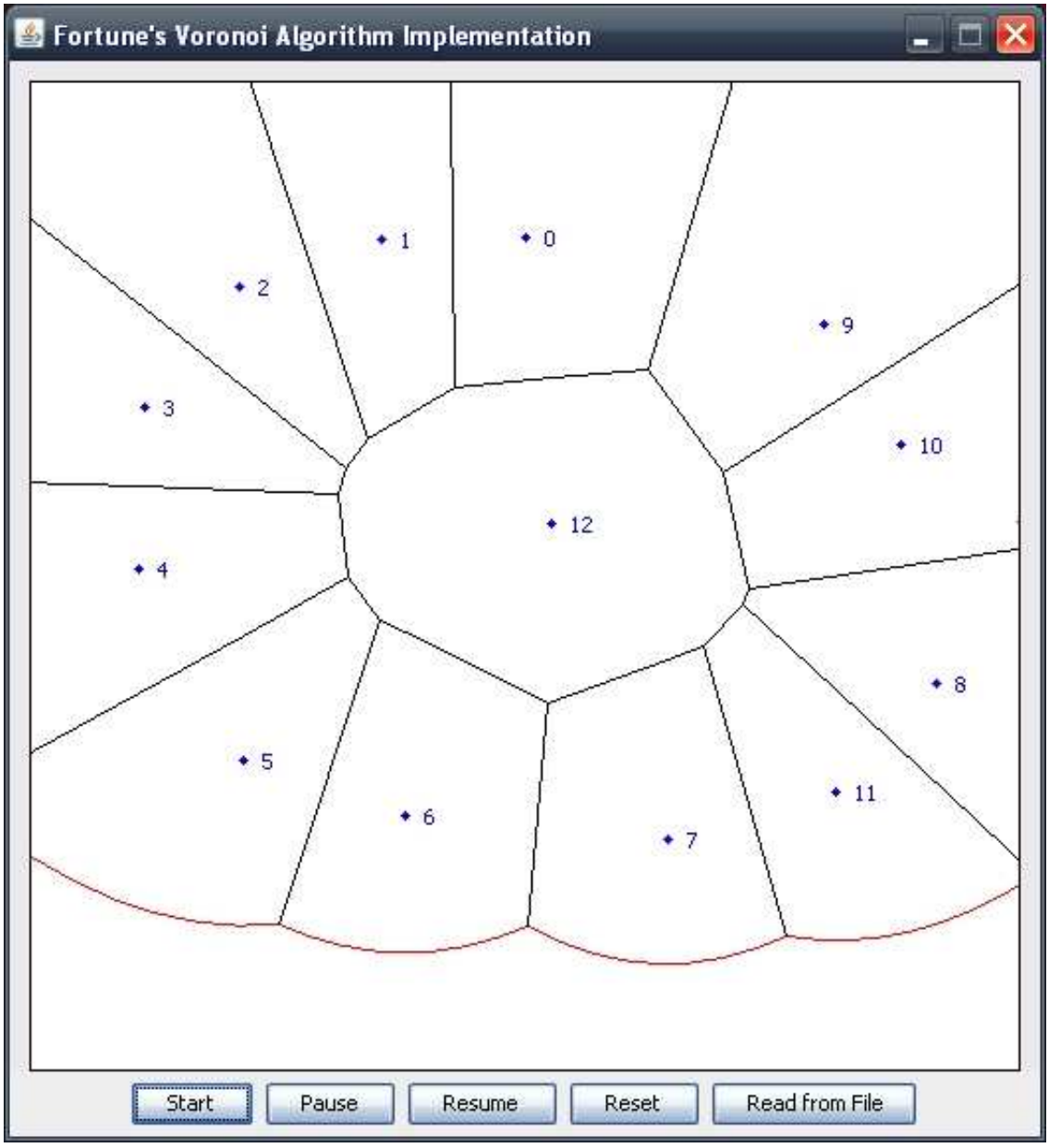
`gen_input.c` : This C program is used to generate inputs for either parts of the assignment. The program takes two inputs, size of the $n \times n$ binary matrix, and the probability of appearance of 1's.

`Voronoi.jar` : This is a self executable program of part 2. It can be run in Unix by typing `java -jar Voronoi.jar` in the terminal or by simply double clicking in Windows.




Fortune's Voronoi Algorithm Implementation
Start
Pause
Resume
Reset
Read from File


Fortune's Voronoi Algorithm Implementation
Start
Pause
Resume
Reset
Read from File


Fortune's Voronoi Algorithm Implementation
0
1
2
3
4
5
6
7
8
9
10
11
12
Start
Pause
Resume
Reset
Read from File