OPTIONAL: Running Project

Geometric Algorithms, Spring 2014

March 24, 2014

Due date: 21th May 2014, before the start of the class.

Problem description

The purpose of this exercise is to compute the intersections of a set of segments of lines or quadratic functions in the plane. The program should read one file which contains two parts:

1. All the input segments of lines or quadratic functions.
2. A list of instructions.

The program should implement a plane-sweep algorithm from left to right. A step means handling the next event (either segment endpoint or an intersection point). The program should handle the next event only as a result of the instruction "step".

Description of the segments. All segments, linear or quadratic, are defined as portions of parametric quadratic curves. That is, $y = at^2 + bt + c$ in the range $[t_1, t_2]$ (for $t_2 \geq t_1$). Obviously, fixing $a = 0$ will make the segment straight. Such a segment occupies one line in the input file in the form

$a\ b\ c\ t_1\ t_2$ (spaces do not matter)

A separate line (in the file) with a single number – the number of segments – precedes all the segment lines. Internally, if there are n segments, the program should name them 0 through $n - 1$.

Description of the instructions. For simplicity, you may assume that instructions are in lower-case only. Some of the instructions should cause the program to print some information to the standard output. The format is critical since an automatic tool will check the correctness of the output. The instructions are:

step: Causes the program to process the next event. In case there are no more events to handle, the program should print:

```
error: no more events
```

step-p: Causes the program to process the next event and to print some information (in one line) in the following C-like format:

```
"event: %1c %6.2f %1d\n"
```

1 character: S (start of segment), E (end of segment), or I (intersection). (one blank)
6 characters: The x coordinate of the event, with two digits after the decimal point. (one blank)
1 character: How many new intersections were found while handling the event. Again, in case there are no more events to handle, the program should print

```
error: no more events
```

status: Prints the status structure in the following C-like format:

```
"status: %d: %d %d ...\n"
```

where the first "%d" stands for the number of segments in the status structure, and the following numbers are the ids of the segments from bottom to top. In case the status structure is empty, the output line should be

```
"status: 0:\n"
```

That is, no extra space after the "0:".

run: Run till the completion of the algorithm. At the termination of the program, the program should print a summary in the C-like format

```
"summary: %d segments, %d intersections\n"
```

(Don't bother to print "segment" and "intersection" in a singular form, even when it is appropriate.)

Notes:

- The implementation of the algorithm and the data structures should be at least as efficient as taught in class. (For example, do not implement a priority queue as a linked list.)
- There is no intention to tackle you with numerical errors and robustness issues. Therefore, no events will be too close, and 2 digits after the decimal point should be enough to hide any numerical inaccuracies. Moreover, no two curves will partially overlap.
- You may assume that no three segments intersect in one point. On the other hand, beware of the fact that segments of quadratic curves may (but don't have to) intersect more than once!
- The output should be written to the standard output, so that it can be redirected to a file and compared to another file.
- Any programming language is acceptable.
- A bonus will be given to programs with a graphic display.
- Despite note #3 above, a special bonus will be given to programs that will be able to handle several events at the same point. To avoid (as much as possible) numerical errors, such events will be only with integral coordinates. To make life easier, no endpoint of one segment will lie on another segment. (That is, only crossing events will be present at the same point in test cases for this bonus.) Also note that if k segments share (in their interiors) the same point, then they induce $\binom{k}{2}$ events!
- You may assume that syntactically the input is always correct.
- Please mention clearly in the documentation (and justify) the space complexity of your implementation.
- Submission instructions: Submit a single .zip file containing all the source files of your implementation and a README file (either .txt or .pdf) in Icorsi. The README file should include: (a) A description of your implementation and its time and space complexity; and (b) A full and clear explanation of how your program should be compiledif necessaryand used (do not send binaries!).

Example

Input file:

```
3
1 0 0 -2 2.5
 -1 0 4 -2.5 2
0 0 3 0 3
step
step-p
step-p
step-p
status
step
step
step
step-p
run
```

Output file:

```
event: S  -2.00 2
event: I  -1.41 1
event: S   0.00 2
status: 3: 0 2 1
event: E   2.00 0
summary: 3 segments, 4 intersections
```