Closest Pair in the Plane

Thore Husfeldt

September 12, 2016

Description

Find a closest pair of points in the plane using divide-and-conquer.

Requirements

As usual, your programme has to take its input either from standard input (STDIN) or from a specified file. It has to write to standard output (STDOUT).

Correctness Your solution must be consistent with the output files in the data directory. However, note that there is a possible source of confusion: Various programming languages support various data types for approximate real numbers. Typically, these types are called `Double` or `Float`. Depending on your choice of internal representation, the approximations will differ between implementations, so you might get a different output than me because of rounding. This cannot be helped; use your good judgement to decide if the error is based on rounding or on a programming mistake.[1]

[1] This is a deliberately messy part of the exercise.

Your code has to be as clear and crisp as possible.

Your solution must run in time $O(n \log^2 n)$, where n is the number of points. If you want, you can implement a slightly more clever version that avoids sorting the input anew for every recursive call and runs in $O(n \log n)$ time. In fact, I found that version just as easy to write.

My solution runs in under a minute on all the inputs in total.

Tips

Have a look at the files – the format is easy enough to parse, but note that the positions of points in the plane is sometimes given as a floating-point number, and sometimes even in scientific notation. Most programming languages have good support for this, *e.g.*, Java's `Double.parseDouble` method is happy to process them. Also, note that the values are sometimes delimited by more than one space character. Again, Java's `String.split` class could be used to break them up. I like to do things by hand, so I used the following regex (in Perl):

```
$number = '[-+]?\d*\.?\d+(?:[eE][-+]?\d+)?';
/(\d+)\s+($number)\s+($number)/;
```

It's probably a good idea to write the naïve quadratic-time algorithm first. It's nice to have access to The Truth for testing, and you need pretty much the exact same code at the bottom of the divide-and-conquer recursion anyway.

Deliverables

1. The source code for your implementation
2. A report in PDF. Use the report skeleton in the `doc` directory.