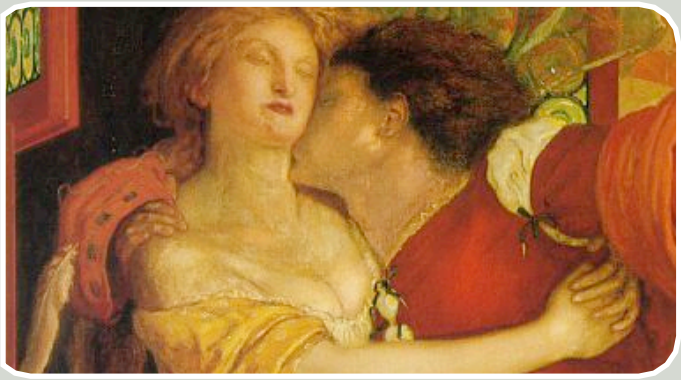
EDAF05 Lab exercise 4

CLOSEST PAIRS IN THE PLANE

Thore Husfeldt
29 April 2009 10:22

Description

Implement a divide-and-conquer algorithm to find a closest pair of points in the plane.



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)/;
```

Another thing: It's probably a good idea to write the naïve $O(n^2)$ 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.

Inputs

The data directory contains a bunch of files called *.tsp taken from the TSPLIB, a library of instances from various sources originally intended for Traveling Salesman algorithms. Some are huge.

I also uploaded six files called "closest-pair-*.in" that contain some really simple instances that you may find useful for testing in the beginning. In each of them, the closest pair is "romeo" and "juliet", and their distance is 1. (You may want to rename the city names to indices for parsing purposes. Do what you want.)

```
NAME : a280
COMMENT : drilling problem (Ludwig)
TYPE : TSP
DIMENSION: 280
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
  1 288 149
  2 288 129
  3 270 133
  4 256 141
...
```

First few lines of a280.tsp

Output

My file "closest-pairs.out" contains the output of my algorithm for all the *.tsp files. It shows, in each line, the name of the input file, the number of points in that file, and the closest distance between those points. I need under a minute to process all of them.

Requirements

Minimal solution

Your algorithm needs to find the right answer for all the "*.tsp" files, using the algorithm described in the book. You are welcome to implement an $O(n \log^2 n)$ version that sorts the sets like S, Q, R anew in every recursive iteration.

Better solution

Implement the "clever" algorithm (*i.e.*, with running time $O(n \log n)$. I found it just as easy to write.

```
NAME: gr96
TYPE: TSP
COMMENT: Africa-Subproblem of 666-city TSP (Groetschel)
DIMENSION: 96
EDGE_WEIGHT_TYPE: GEO
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
 1 14.55 -23.31
 2 28.06 -15.24
 3 32.38 -16.54
 4 31.38 -8.00
...
```

First few lines of gr96.tsp

```
...
8 7.21100e+03 1.19020e+04
9 1.53280e+04 7.87600e+03
10 1.05760e+04 5.21400e+03
11 1.25600e+04 2.42000e+03
12 1.63680e+04 4.31200e+03
...
```

Some lines of rl1849.tsp