Interval Partitioning

Troels Bjerre Sørensen and Riko Jacob

August 27, 2015

Description

Implement the greedy algorithm for interval partitioning (p. 166 in Kleinberg and Tardos) and run it on several test inputs.

Files

The files for this exercise are on LearnIT.

Input

A typical input file is shown below. The first line consists of an integer, called n, giving the number of intervals (requests) to schedule. Then there is an empty line. Finally, there are n lines consisting of the two integers that form the interval.

```
5

5 7
3 8
2 4
1 5
5 8
```

Output

The first line of the expected output consists of an integer, k, being the minimal number of partitions required for the scheduling. Then there is an empty line. Finally, there are n lines, each with 3 integers; the two first describe the interval while the last describes which partition the interval has been assigned to as a number between 0 and $k - 1$. The intervals must be presented in the same order as in the input, but any numbering of the partitions is valid.

```
3

5 7 0
3 8 2
2 4 1
1 5 0
6 8 1
```

Requirements

Language Java.

Infrastructure Your program has to take its input either from standard input (STDIN) or from a specified file. It has to write to standard output (STDOUT). Assuming all the files are in the right place, we must be able to run the program from the command line with at least one of the following two commands:

```
% java IP < part-1.in
% java IP part-1.in
```

(Your're welcome to write your code in Eclipse or whatever rocks your boat. But the above has to work.)

Correctness The course data file directory (linked from the course home page) contains a number of pairs of files called "ip-*.in" and "ip-*.out" that contain matching input and expected output. Your solution must be consistent with these files, in that you must produce a solution using the same number of partitions, though the assigned partitions may differ.

Solution quality

Minimal, acceptable solution Implement the greedy algorithm so that it works correctly. Especially, your program shall give a correct answer on all test cases. The running time of your implementation has to be that of sorting. You are welcome to use a library call to do the sorting, in which case you should argue that everything else takes at most $O(n \log n)$ time. In particular, your running time and space usage may not depend on the universe size, i.e., the numbers used to denote the endpoints of the intervals.

Deliverables

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

Tips

- Algorithmically, this is an easy exercise. The algorithm is simple and spelled out in detail on p. 166 in the book (in the international edition).