CS25100: Data Structures and Algorithms, Fall 2017

Project 3

Handed out: **October 2, 2017**
Due: **October 16 at 11:59pm**

---

examples.tgz
stdlib.jar
algs4.jar

Project Outline

The input consists of a set `S` of `N` two-dimensional records of integers, each of which consists of a pair of coordinates `(x[i],y[i])` for `0 ≤ i ≤ N-1`, where the `x[i]` are distinct (i.e., no two are equal) and the `y[i]` are also distinct.

Intuitively, the x coordinates represent one desirable attribute of each record, whereas the y coordinates represent another (different) desirable attribute of the record.

Part 1

Overview

Write a program that eliminates from `S` every record `(x[i],y[i])` for which there exists another record `(x[j],y[j])` having both `x[j] > x[i]` and `y[j] > y[i]`.

Intuitively, we want to eliminate all records that are doubly inferior to at least one other record, where "doubly" means in terms of both x coordinate and in terms of y coordinate.

Output the surviving ("filtered") records in `S` sorted according to increasing order of their x coordinates.

Note: You are allowed to use existing sorts (no need to write your own).

Implementation

Create a class `Filter.java` to perform the task described above. Your program should process input using `StdIn.java` in the following format.

The first line consists of a single integer `N`, the number of records to read. This is followed by a list of `N`

records, one per line, each consisting of two integers `x` and `y` separated by a space. The records do not appear in any specific order.

Your program should output the filtered, sorted records using `StdOut.java`. Write one record per line, `x` followed by `y` separated by a space.

We have provided example input / output files in the archive linked at the beginning of the handout. Use `filter1.txt`, `filter2.txt`, and `filter3.txt` as input, and `filter1out.txt`, `filter2out.txt`, and `filter3out.txt` as output to test your program.

Part 2

Overview

Put the sorted input `N` records (unfiltered) into an array, sorted by their `x` coordinates in increasing order. That array can be viewed as representing a complete binary tree `T`. For every node `v` of `T`, create an array `L[v]` that contains the records in the subtree of `v` in `T`, sorted according to their y coordinates. Note that this creates multiple copies of a record (as many as ther record's ancestors in `T`).

Hint: Create the `L[v]`'s in bottom-up order, so that you can obtain the `L[v]` of a `v` whose children are `u` and `w` by merging the already-computed `L[u]` with the already-computed `L[w]` and with `v`.

Use the tree `T` and the `L[v]` lists to efficiently process queries of the type `Q(a,b)` that outputs the records whose `x` coordinate is greater than `a` AND y coordinate is greater than `b`. The records output from each query should be sorted by their `x` coordinates in increasing order.

Performance for each such query should be $O((\log N)^2 + m \log m)$ time where `m` is the number of output records.

Note: $O(\log N + m \log m)$ time performance is possible, but not required.

Implementation

Create a class `Query.java` to perform the task described above. Your program should process input using `StdIn.java` in the following format.

The first part of the input is the same as for part 1: the first line has a single integer `N`, followed by `N` records, one record per line, each consisting of two integers `x` and `y` separated by a space. As in part 1, the records do not appear in any specific order. The line following the last record contains a single integer `Q`, the number of queries to process. This is followed by `Q` lines, one query per line, consisting of two integers `a` and `b` separated by a space. Your program should process and output each query in the order it appears.

Your program should print the records output from each query to `StdOut.java`. For each query, output `m` records, one per line, each consisting of two integers `x` and `y` separated by a space, where `m` is the number of records returned by that query. If a query returns zero records, output a single line with the string "`none`" (without quotes).

As with part 1, we've included example input / output files in the archive linked at the top of the handout. Use `query1.txt`, `query2.txt`, and `query3.txt` as input, and `query1out.txt`, `query2out.txt`, and `query3out.txt` as output to test your program.

Note: since we require each query to have a specific complexity, your code will be timed to ensure this requirement is met. Specific time limits are yet to be determined.

Submission

Submit your solution on or before **Monday, October 16, 11:59pm**. The submission procedure is the same as for previous projects. Inside your working directory for this project on data (e.g. `~/cs251/project3`), create a folder in which you will include all source code used and libraries needed to compile and run your code. Specifically, your submission must include the following files:

- `Filter.java`
- `Query.java`
- `readme.txt` (if needed)

DO NOT use absolute paths in your files since they will become invalid once submitted. Optionally, you can include a README file to let us know about any known issues with your code (like errors, special conditions, etc).

After logging into `data.cs.purdue.edu`, please follow these steps to submit your assignment:

- Enter the working directory for this project

```
%cd ~/cs251/project3
```

- Make a directory named `<your_first_name>_<your_last_name>` and copy all the files needed to compile and run your code there.
- While still in the working directory of your project (e.g. `~/cs251/project3`) execute the following turnin command:

```
%turnin -c cs251 -p project3 <your_first_name>_<your_last_name>
```

- Keep in mind that old submissions are overwritten with new ones whenever you execute this command. You can verify the contents of your submission by executing the following command:

```
%turnin -v -c cs251 -p project3
```

- Don't forget the `-v` flag here, as otherwise your submission would be replaced with an empty one.