Linköping University
Department of Science and Technology/ITN
Aida Nordman

TND004: Data Structures
Lab 3: part 2

Goals

- To use sorting in a practical problem to improve efficiency.
- To analyse algorithms in terms of their time and space complexity.
- To motivate objectively the choices made and relate to the known scientific results in the field.

Getting started

This exercise is about improving the given implementation of a simulation system for particles collision.

To get started with the exercise, perform the steps indicated below.

- Create a folder for this lab named e.g. `Lab3-part2`.
- Download the zipped folder with the files for this exercise from the course website and unzip it into the lab folder.
- Execute the remaining instructions in the file `README.md` (can be opened with Notepad). The instructions given in `README.md` are similar to the ones given for the lab3-part1.
- Compile, link, and execute the program (the `main` is in `lab3-part2.cpp`). When prompted for a points file, enter e.g. `points200.txt`. All points in the input file are then plotted. Note that this program only provides plotting functionality, for both input lines and line segments[1].
- Read the remaining lab description, exercise 1 and exercise 2.
- Reviewing lecture 8.
- Plan your code and implement the line segments discovery system using the presented sorting based-algorithm. Contact the course responsible if you want to have a "*template*" CMake file for your code.

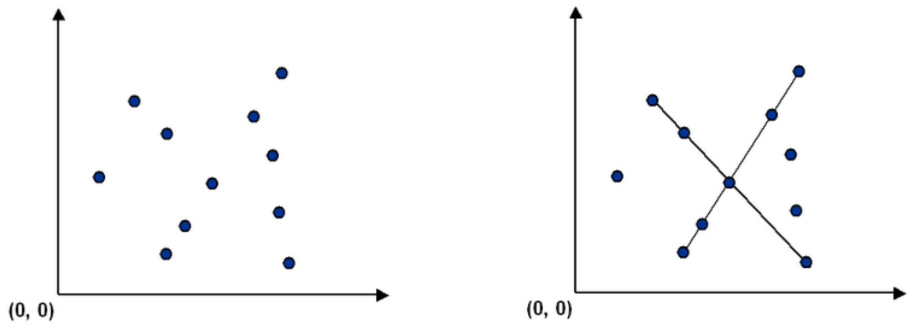
If you have any specific question about the exercises, then send us an e-mail. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code to the e-mail's subject, i.e. "`TND004:` …".

Exercise 1: To find patterns in large datasets of points

Data about the real world is often collected as sets of points. An interesting problem is how the real-world objects that produced those points can be reconstructed.

[1] At the start of this exercise, there are no line segments yet discovered from the points files. Thus, no line segments are plotted, yet.

One possible way to tackle the problem above is to find every line segment that connects 4 or more distinct points belonging to a given dataset of $n > 0$ 2D-points, as illustrated below.



A simple brute force approach could be used to solve this problem. This strategy considers four points at a time and checks if they all lie on the same line segment.

```
for (i1 = 0; i1 < size(points) - 3; ++i1) {
    for (i2 = i1 + 1; i2 < size(points) - 2; ++i2) {
        for (i3 = i2 + 1; i3 < size(points) - 1; ++i3) {
            for (i4 = i3 + 1; i4 < size(points); ++i4) {
                if (onSameLine(points[i1], points[i2], points[i3], points[i4])) {
                    // …
                }
            }
        }
    }
}
```

You can easily see that the brute force approach requires $O(n^4)$ time. In practice this algorithm would not be feasible for even medium size datasets. It is possible to solve the problem much faster than the brute force solution. The algorithm is presented in the next section.

A sorting-based algorithm

You should implement the following algorithm which involves sorting. Given a point p, the following method determines whether p belongs to a set of 4 or more collinear points.

1. For each other point q, consider the slope of the line segment that connects point p with q[2].
2. Sort the points according to the slope with point p.
3. Check if any 3 (or more) adjacent points in the sorted order have the same slopes with p. If so, these points, together with p, are collinear and form a line segment that connects 4 or more distinct points.

   Note that the sorting places together the points that have the same slope with respect to point p.

Feel free to use containers and algorithms available in the Standard Template Library of C++.

[2] The line defined by the points $p = (x_1, y_1)$ and $q = (x_2, y_1)$ has slope $\frac{(y_2-y_1)}{(x_2-x_1)}$.

Requirements

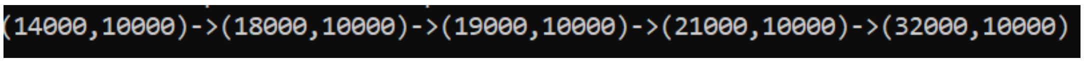
Your program must satisfy the following requirements.

- Run in $O(n^2 \log n)$ time.
- Output a **non-redundant** list of the line segments (with four or more points) discovered. Each displayed line segment should include all points in the input that belong to it[3]. This list can be written to `std::cout` or to a file.

The output list is considered non-redundant if the following two criteria are met.

- The output list contains only one representation of each line segment (e.g. different permutations of the same points should be excluded).
- Subsegments of a line (or of one of its permutations) should not be part of the output, either.

For example, consider the data points in the input file `points6.txt`. Then, the program should only output the following line.



```
(14000,10000)->(18000,10000)->(19000,10000)->(21000,10000)->(32000,10000)
```

Including any of the following in the output list would have made the output list redundant.

```
(14000,10000)->(18000,10000)->(19000,10000)->(21000,10000)
(18000,10000)->(19000,10000)->(21000,10000)->(32000,10000)
(19000,10000)->(32000,10000)->(21000,10000)->(14000,10000)->(18000,10000)
```

Input data files

Several input files with 2D-points are provided in the folder named `detectionsystem/data`. These files contain points coordinates which are integers in the interval $[0, 32767]$. Each input file starts with the number of points ($n > 0$) followed by the points coordinates, one point per line.

For each input file, the expected output (i.e. discovered line segments) is available in the folder `detectionsystem/data/output/lines-discovered`. The points forming the line segments are sorted first by the y-coordinate and then by the x-coordinate (e.g. point $(14080, 12032)$ appears before point $(13056, 13056)$, while point $(13120, 20224)$ appears before $(14080, 20224)$).

To plot the discovered line segments

To be able to plot the discovered line segments for an input file of points (e.g. `points6.txt`), the following is required.

- Your program must also generate a text file with the line segments discovered. This file should consist of a pair of points coordinates per line (i.e. four integers). These line coordinates correspond to the coordinates of the start and end points of a line segment, respectively (see example below).

```
10000 0 0 10000
3000 4000 20000 21000
```

[3] Just displaying the start and end points of a line segment is not enough.

- The line segments file should be named after the input file of points. For instance, if the input file is named `points6.txt` then the line segments file must be named `segments-points6.txt`.
- The line segments files should be placed in the folder `detectionsystem/data/output`. You can then execute the provided program and it will plot both the input points and the line segments discovered in the input.

In the folder `detectionsystem/data/output/line-plots`, you can also find `png` files with the plot of the line segments discovered from the corresponding points files, with exception for the input file `largeMystery.txt`.

Exercise 2: complexity analysis

For this exercise, you are required to write a concise, clear, and well-motivated analysis of your solution. The following must be addressed in your report.

- Any assumptions made.
- Identify the variable(s) on which your analysis depends on.
- Time complexity (see requirements).
- Space complexity.

Presenting lab and deadlines

The exercises in this lab are compulsory and you should demonstrate your solutions during the lab session *Lab3 RE*. Read the instructions given in the labs webpage and consult the course schedule.

Necessary requirements for approving your lab are given below.

- Present your implementation for the sorting-based algorithm. You need to clearly describe how the requirements are met.
- The code must be readable, well-indented, and use good programming practices.
- Discuss with the lab assistant the complexity analysis. Remember to bring this part also written. The lab assistant may require you to hand-in the analysis part for further feedback. Do not forget to indicate the name plus LiU-id of each group member. Unreadable answers will be rejected.
- Answer the question: what pattern is hidden in the points file `largeMystery.txt`?

If your solutions for the exercises in lab3/part2 have not been approved in the scheduled lab session *Lab3 RE* then it is considered a late lab. Late labs can be presented provided there is time in a another RE lab session. All groups have the possibility to present one late lab on the extra RE lab session scheduled in the end of the course.