Long Take-Home Exam 4

(Deadline: 21th of June 2023, Wednesday, 23:59)

1 Problem

You are given n line segments on the Cartesian plane with the following properties:

- All segments have a positive length.
- No segment is vertical.
- All segments are disjoint, with no common points at all.

You are asked which two segments have the *minimum vertical distance* between them. The vertical distance between two segments A and B is the length of the shortest vertical line segment that connects a point of A to a point of B. Naturally, such a distance is only defined if A and B occupy at least one common x-coordinate.

To ease your job for this problem, we also guarantee that:

- There will be at least one pair of segments occupying the same abscissa (x-coordinate).
- All of the $2n$ endpoints of the segments have unique abscissae.
- The pair with the minimum vertical distance is unique.

2 Input Format

$$\begin{array}{l}\texttt{N}\\ \texttt{W}_1\ \ \texttt{L}_1^x\ \ \texttt{L}_1^y\ \ \texttt{R}_1^x\ \ \texttt{R}_1^x\\ \ldots\\ \texttt{W}_N\ \ \texttt{L}_N^x\ \ \texttt{L}_N^y\ \ \texttt{R}_N^x\ \ \texttt{R}_N^y\end{array}$$

$\texttt{N}$ = The number of segments.
$\texttt{W}_i$ = A unique word identifying the ith segment.
$\texttt{L}_i^x$ = The abscissa of the left endpoint of the ith segment.
$\texttt{L}_i^y$ = The ordinate of the left endpoint of the ith segment.
$\texttt{R}_i^x$ = The abscissa of the right endpoint of the ith segment.
$\texttt{R}_i^y$ = The ordinate of the right endpoint of the ith segment.

3 Output Format

$$\texttt{W}_{above}\ \ \texttt{W}_{below}$$

$\texttt{W}_{above}$ = The word identifying the *higher* of the two segments with the minimum vertical distance.
$\texttt{W}_{below}$ = The word identifying the *lower* of the two segments with the minimum vertical distance.

4 Limits

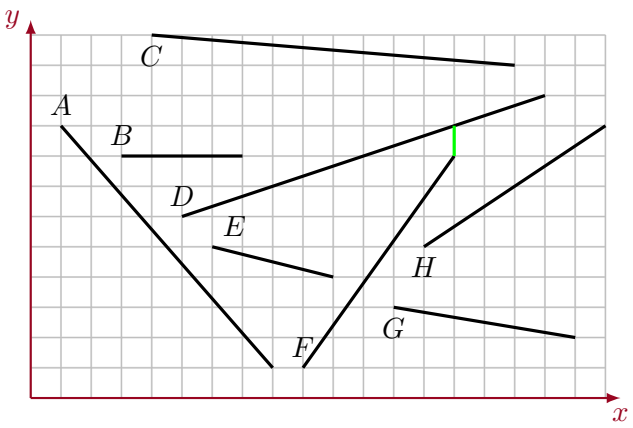
$1 \leq \texttt{N} \leq 100\,000$.
All identifying words are alphanumerical and at most 8 characters.
All coordinates are integers in the range $[1, 2\,000\,000\,000]$.
Time Limit: 1.5 second, Memory Limit: 256 MB, Stack Limit: 8 MB

5 Clarifications

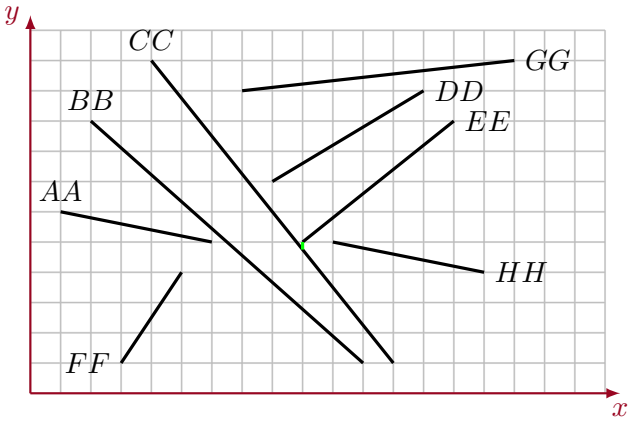
- Your solution is expected as a C++ program source named `1the4.cpp` that reads from the standard input and writes to the standard output.
- It is OK to copy code from the sample codes we shared in our course website in ODTÜClass. You can also copy code from your previous THE submissions for this course. Copying from elsewhere will be considered cheating.
- You are supposed to submit your code via the VPL item in ODTÜClass. Use the "evaluate" feature to run the auto-grader on your submission.
- The grade from the auto-grader is not final. We can later do further evaluations of your code and adjust your grade. Solutions that do not attempt a "reasonable solution" to the given task may lose points.
- We will compile your code with `g++` using the options: `-std=c++2a -O3 -lm -Wall -Wextra -Wpedantic`
- Late submissions are not allowed.

6 Sample Input/Output Pairs

Input	Output
8 A 1 9 8 1 B 3 8 7 8 C 4 12 16 11 D 5 6 17 10 E 6 5 10 4 F 9 1 14 8 G 12 3 18 2 H 13 5 19 9	D F # Abscissa: 14 # Distance: 1.00



Input	Output
8 AA 1 6 6 5 BB 2 9 11 1 CC 4 11 12 1 DD 8 7 13 10 EE 9 5 14 9 FF 3 1 5 4 GG 7 10 16 11 HH 10 5 15 4	EE CC # Abscissa: 9 # Distance: 0.25



7 Hints

- A well implementation of the $O(n \log n)$-time plane-sweep algorithm that we described in the class is expected to get full points.
- As usual, if you cannot code the aforementioned algorithm, please code a slower one to get partial points.

- Using a `float` for coordinates should be sufficient for this problem; however, if you are suspicious, feel free to use a `double` or even `long double`.
- The standard library classes `std::(multi)set` and `std::(multi)map` are red-black tree (balanced binary search tree) implementations. You can provide a *custom comparison* ($<$) function to be used by the implementation. For instance:

```
auto myCompare = [](const MyObject *first, const MyObject *second)
{
    return first->myField < second->myField;
};

std::set<const MyObject *, decltype(myCompare)> mySet(myCompare);
```

- Your comparison function needs to be *informed about the sweepline position*. You can do it in various ways. Two possibilities:
  - Use a global variable.
  - Capture a local variable by reference in your lambda expression.
- You can use `lower_bound` and `upper_bound` member functions to do predecessor and successor queries on your red-black tree. However, read the function documentations carefully. The functions do not behave as expected by intuition. (To start with, they are not symmetric of each other). You will have to compare the returned iterator against `begin()` and/or `end()` to check whether a predecessor/successor exists. You may also need to modify the returned iterator (via $++$ or $--$) to reach the predecessor/successor.
- As a side exercise, imagine how you could adapt your solution if it was possible that the endpoints were allowed to have the same x-coordinate. This kind of edge cases are known as *degeneracies* in computational geometry. Unfortunately, they are frequently faced in geometric programming, usually in worse forms, and are tedious to deal with.

Good luck in the finals and have a nice holiday!