View spot finder

There is a map excerpt of a hilly landscape. We call it a mesh. The mesh is partitioned in triangles; we call them elements. For each element a scalar value is assigned, which represents the average spot height in this triangle as compared to the sea level.

Mesh definition: A mesh is a collection of elements and nodes. Each node is a location on the map, given as a 2-dimensional point. It has an identification number (ID), two coordinates and can serve as a vertex for an element. Every element has an ID and is defined by three vertices – by three node IDs. The mesh can be bulky as depicted below:

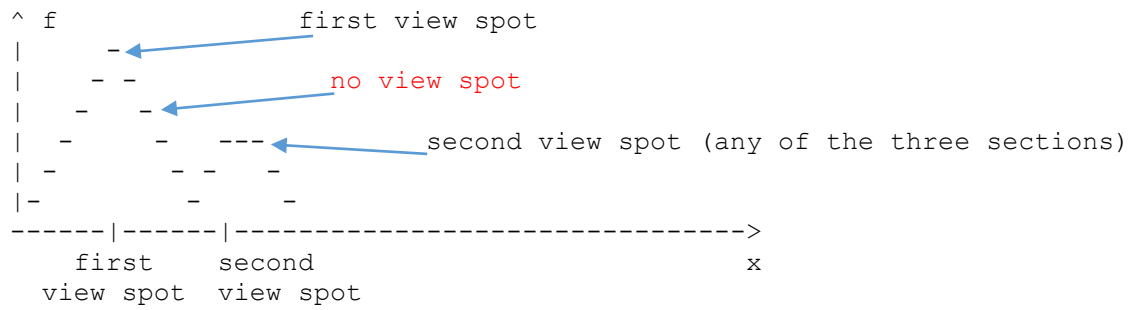
Problem

For a walking tour we would like to identify the view spots. A view spot is the element where the height reaches its local maxima, that is, all the neighboring elements are not higher. We consider two elements as neighbors if they share at least one node – vertex.

The task is as follows: Given a mesh and an integer number N, find the first N view spots ordered by the spot height starting from the highest to the lowest.

In the case when two or more neighboring elements have exactly the same value, only one of the elements should be reported as a view spot.

In the case of 1-dimensional mesh, where we have a track sections instead of triangles, the idea is illustrated on the picture below.



Input

1. A JSON file with the mesh and the height values. The file contains three sections: `nodes`, `elements`, and `values`. The syntax is as follows:

```
{
  nodes: [
    {id: node_id1, x: <number value>, y: <number value>},
    {id: node_id2, x: <number value>, y: <number value>},
    {id: node_id3, x: <number value>, y: <number value>},
    ...
  ],
  elements: [
    {id: element_id1, nodes: [node_id1, node_id2, node_id3]},
    ...
  ],
  values: [
    {element_id: element_id1, value: <number value>},
    ...
  ]
}
```

   An example of a mesh file is attached.

2. Integer number `N` that defines how many view spots must be found and written out.

Output

List of N view spots (view spot element ID, height value on this element) ordered by value from the highest to the lowest. The syntax is as follows:

```
[
  {element_id: element_id1, value: <number value>},
  ...
]
```

Messages

If a message (error/warning/information) makes sense in some situation, it can be written into stderr or into a log file but not into the standard output. The standard output shall only contain the list of hotspots formatted as above.

Interface

The program should have a simple command line interface as follows:

```
[program name].exe <mesh file> <number of view spots>
node [program name].js <mesh file> <number of view spots>
java -jar [program name].jar <mesh file> <number of view spots>
python [program name].py <mesh file> <number of view spots>
```

Performance requirements

- The program shall be able to find all view spots in a mesh with 10000 elements in less than 15 seconds, ideally under 1 second. The time assumes an average laptop without any parallelization and that the script is written in Python. The Java, JavaScript or C# implementations shall be around twice as fast.
- The calculation time of a mesh with 20000 elements shall be less than 3 times higher than the calculation of a mesh with 10000 elements.

Attachments

- `mesh.json` – a small example file for quick testing
- `mesh_x_sin_cos_10000.json` – an example mesh with 10000 elements
- `mesh_x_sin_cos_20000.json` – an example mesh with 20000 elements