Instructions

Please complete both problems and enter your solutions into the appropriate files. You may use the included unit tests to verify your solution. To submit the completed test, attach the relevant source files to an email to your recruiter. See readme.md for more information on the files of interest.

Problem #1: Visitor Sessions

Your website is getting popular and you would like to get more insights about your traffic. You want to know how many visits, unique visitors, and sessions you are serving.

Definitions

- Visits – the number of times a site is viewed
- Unique visitors – the number of distinct individuals viewing pages from the website
- Sessions – the number of times a visitor is active on the website, according to a 30-minute inactivity deadline. Once a visitor has been inactive for 30 minutes the session is ended, and a subsequent visit starts a new session. So for example, if a visitor views the site at 0:00, 0:20, 0:40, 1:20, and 1:30, these five visits would be considered two sessions.

Input

You will receive an array of `WebsiteVisit` objects. The `WebsiteVisit` objects are sorted chronologically, from oldest to most recent.

`WebsiteVisit` Object Specification

- `visitorId`: a unique identifier for a visitor.

- `timestamp`: a timestamp in seconds

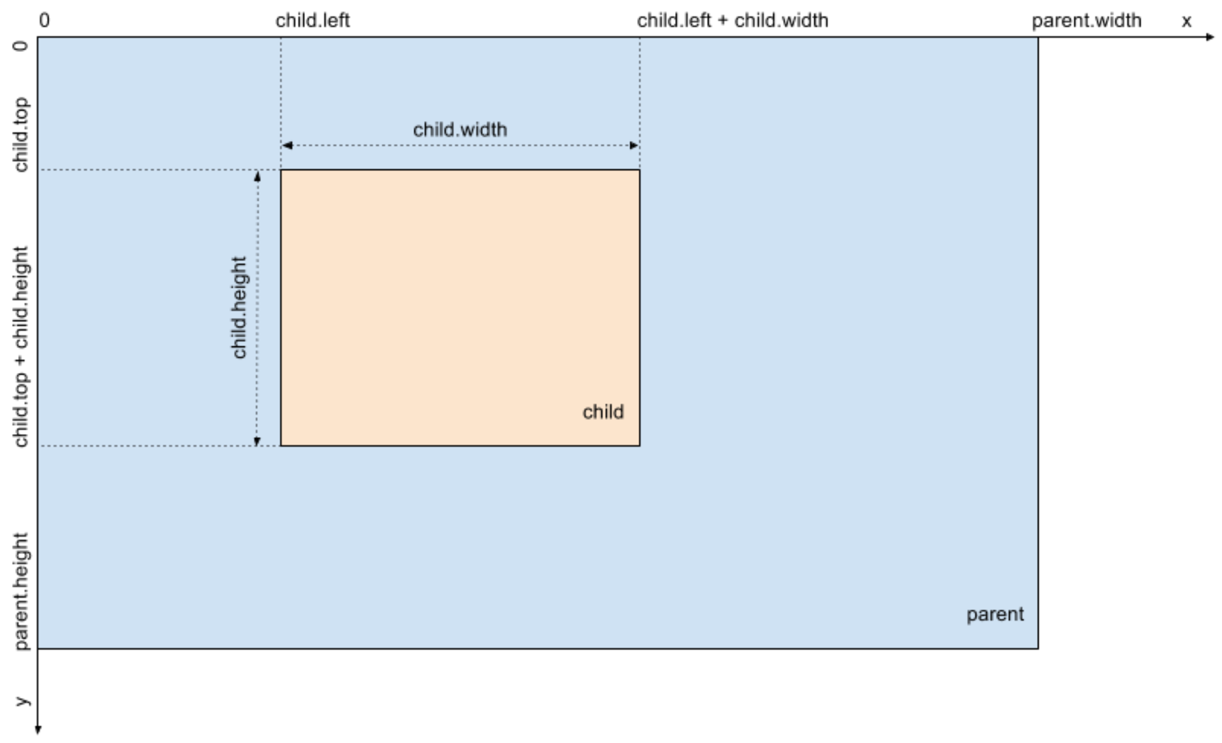
Output

You will return the following values:

- number of website visits
- number of unique visitors
- number of sessions

Problem #2: Template Framework

You are working with a UI rendering framework that represents a page as a tree of rectangle nodes. Rectangles nest within each other to form a complex rendering of rectangles (like UIViews in iOS). They can nest several levels down.
Just as in iOS, it works in a two-dimensional coordinate system where the upper-left corner has the coordinates $(0, 0)$. The **x** axis increases towards the right and the **y** increases towards the bottom.



Input

You will receive a `rootNode` and a `CGPoint`.

Node Object Definition

- `nodeId`: a unique identifier. The id of a node is an opaque string with no intrinsic significance. You should not use the id to make any inferences about it.
- `frame`: the frame of the node, relative to its parent origin. All values (x, y, width, height) will be ≥ 0
- `children`: an array of nodes, which are the children that are nested within this node

Constraints

- Children are strictly contained in their parent (no overflow). This means that:
  - $0 < (child.left + child.width) \leq parent.width$
  - $0 < (child.top + child.height) \leq parent.height$

- Children nodes may overlap each other. In this case, the child with the higher index covers the child with the lower index.
- Children may not completely fill a parent.

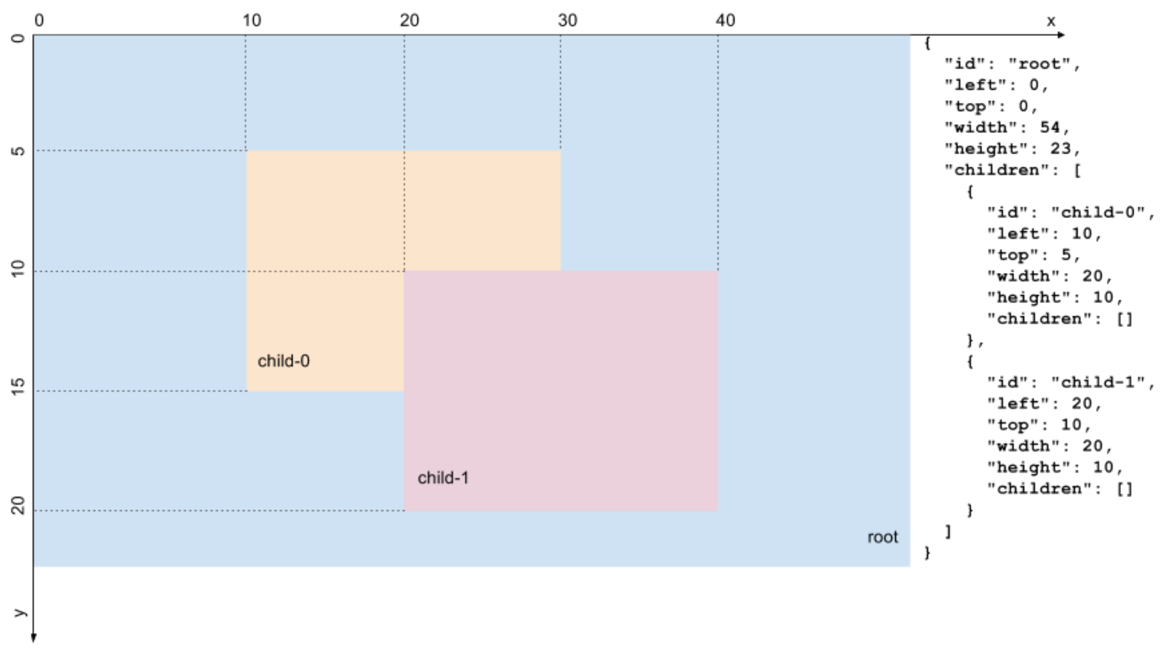
Output

Find the Node that is drawn at the point's coordinates.
Return an array of IDs describing the path from the root Node to the Node that you found.
If there is no match, return an empty array.

Reminder

If two children overlap, the one with the higher index in the `children` array is chosen.
Also, keep in mind that a child node can have child nodes of its own.

Example



```
{
  "id": "root",
  "left": 0,
  "top": 0,
  "width": 54,
  "height": 23,
  "children": [
    {
      "id": "child-0",
      "left": 10,
      "top": 5,
      "width": 20,
      "height": 10,
      "children": []
    },
    {
      "id": "child-1",
      "left": 20,
      "top": 10,
      "width": 20,
      "height": 10,
      "children": []
    }
  ]
}
```

In the example above, for pixel coordinates $(22, 12)$, the pixel falls into the bounds of "child-0" and "child-1" but "child-1" appears last in the children array. You should thus return the path from "root" to "child-1" as ["root", "child-1"].